

1N-61

48767

P. 129

Computer Integrated Documentation

Guy Boy

(NASA-TM-103870) COMPUTER INTEGRATED
DOCUMENTATION (NASA) 129 p CSCL 09B

N92-11673

Unclas
G3/61 0048767

September 1991

Working Paper – This Technical Memorandum is an unedited report of research that is ongoing. It is being released in this format to quickly provide the research community with important information.

NASA

National Aeronautics and
Space Administration

Computer Integrated Documentation

Guy Boy, Ames Research Center, Moffett Field, California

September 1991



National Aeronautics and
Space Administration

Ames Research Center

Moffett Field, California 94035-1000

Contents

Summary.....	1
Acknowledgements.....	1
1. Description of the Problem.....	3
1.1. Introduction.....	3
1.2. Intelligent Assistant Systems Background.....	4
1.2.1. Motivations for Designing Intelligent Assistant Systems.....	4
1.2.2. Human Operator Modeling.....	5
1.2.3. Situation Recognition Analytical Reasoning Model.....	5
1.2.4. Knowledge Representation Issues.....	6
1.2.5. Associated Human Factors Problems.....	7
1.3. Documents Management and Maintenance.....	9
1.3.1. Introduction.....	9
1.3.2. Technical Documentation from Design to Operations.....	10
1.3.3. Navigation Problems in Documentation.....	12
1.3.4. Context.....	12
1.3.5. Knowledge-Based Indexing.....	13
2. Possible Technologies.....	15
2.1. Conventional Approaches to Indexing and Information Retrieval.....	15
2.1.1. Definitions.....	15
2.1.1.1. Indexing.....	15
2.1.2.1. Frequency-Based Methods.....	16
2.1.2.2. Descriptor-Discrimination Value.....	18
2.1.2.3. Conclusion.....	18
2.1.3. Compound Descriptor Formation.....	19
2.1.4. Thesaurus Generation.....	19
2.1.4.1. Enlarging Pattern-Matching Capabilities.....	20
2.1.4.2. Aliases.....	20
2.1.4.3. Conclusion.....	20

2.1.5. Some Information Retrieval Models.....	20
2.1.5.1. Vector Space Model.....	21
2.1.5.2. Automatic Referent Classification.....	23
2.1.5.3. Probabilistic Retrieval Model.....	29
2.1.5.4. Fuzzy Set Retrieval Model.....	31
2.1.6. Conclusions.....	37
2.2. HyperText.....	37
2.2.1. History and Definitions.....	37
2.2.1.1. Linear text.....	37
2.2.1.2. Nonlinear text.....	38
2.2.1.3. What a HyperText system is not	38
2.2.1.4. Browsing.....	39
2.2.1.5. Implementation	39
2.2.2. From Text to Hypertext.....	43
2.2.2.1. Description of an example of paper documentation	43
2.2.2.2. Modelling the nodes and the links.....	45
2.2.3. Conclusions.....	46
2.3. Knowledge-Based Systems.....	46
2.3.1. Concept Indexing.....	47
2.3.1.1. Concept hierarchies acquisition from documentation experts.....	47
2.3.1.2. Concept indexing by end-users.....	47
2.3.2. Text Analysis and Interpretation	48
2.3.3. Conclusions	49
2.4. Synthesis.....	49
3. Contextual Knowledge.....	51
3.1. Representing Contextual Knowledge.....	51
3.1.1. Introducing the Block Representation.....	51
3.1.2. Context in Information Retrieval.....	54
3.1.2.1. External vs. internal context	54
3.1.2.2. Current context and context patterns.....	54
3.1.2.3. Mutually-independent vs. dependent referents within a block	54
3.2. Using Contextual Knowledge	55

3.2.1. Block Management.....	55
3.2.2. Using Context in Information Retrieval.....	56
3.2.2.1. Context compilation.....	56
3.2.2.2. Compiled context use.....	58
3.2.2.3. Using abnormal conditions	58
3.3. Context-Sensitive Indexing	59
3.3.1. Building Suboptimal Block Bases by Semi-Automatic Indexing.....	59
3.3.1.1. Extracting words and their frequency.....	59
3.3.1.2. Extracting single-term descriptors.....	59
3.3.1.3. Constructing compound descriptors	59
3.3.1.4. Constructing aliases.....	60
3.3.1.5. Building descriptor-referent links	60
3.3.2. Intentional vs. Experimental Search.....	60
3.3.2.1. Definitions	60
3.3.2.2. Detecting the user search mode.....	61
3.3.3. Success and Failure Feedback	61
3.3.4. User-Guided Indexing in Experimental Search Mode.....	61
3.3.4.1. Recording a trace.....	62
3.3.4.2. Describing a successful referent.....	62
3.3.5. User-Guided Indexing in Intentional Search Mode.....	62
3.3.5.1. Discovering Abnormal Conditions	62
3.3.5.2. Refining Index-Blocks by Context.....	64
3.3.5.3. Example of Application.....	65
3.3.6. Conclusions	66
4. Implementation of a Computer Integrated Documentation System	69
4.1. Space Station Program Requirement Document Application.....	69
4.1.1. Introduction	69
4.1.2. Analysis of the Program Definition and Requirement Document.....	70
4.1.3. The Technical and Management Information System	70
4.2. The Computer Integrated Documentation System.....	71
4.2.1. HyperText Database.....	71
4.2.1.1. General infrastructure of referents	71
4.2.1.2. Referents	72
4.2.1.3. Descriptors.....	74

4.2.2. Knowledge-Based Management and Maintenance System	74
4.2.2.1. Descriptor agenda	74
4.2.2.2. Access to a compound descriptor (triggering condition of a block) from a single-term descriptor.....	75
4.2.2.3. Representation of referents.....	75
4.2.3. User Interface	79
4.2.3.1. Control panel	79
4.2.3.2. Basic stacks (windows)	79
4.2.3.3. Various kinds of referents.....	79
4.2.3.4. Visual aids	82
5. Theoretical Considerations	85
5.1. Navigation in HyperSpace	85
5.1.1. Theory of Navigation in Hyperspace.....	85
5.1.1.1. User's search mode and ontology.....	85
5.1.1.2. The block representation as a navigation aid.....	86
5.1.2. Hypertext Metalevel	87
5.2. Acquisition of Indexing Knowledge.....	87
5.2.1. Semantic Indexing	87
5.2.2. Extracting Blocks from Traces in the Hyperspace.....	88
5.2.2.1. Analysis of user's traces	88
5.2.2.2. User interface capabilities for extracting useful referents-descriptors relations	89
5.2.3. Context Clustering.....	89
5.2.3.1. Placing a context pattern in an existing class.....	90
5.2.3.2. Creating a new class	90
5.2.3.3. Merging several classes into a single class.....	91
5.2.3.4. Splitting a class into several classes.....	92
5.3. Generation and Maintenance of a Large Documentation.....	92
5.3.1. Generation of Descriptors	93
5.3.1.1. Generation of descriptor at the user level.....	93
5.3.1.2. Maintenance of a descriptor dictionary	93
5.3.2. Incremental Reinforcement from User's Feedback in Context.....	94
5.3.3. Semantic Correlation between Documents.....	95
5.4. Relation to Other Work.....	97
5.4.1. Regarding Blocks as Procedures	97

5.4.2. Intelligent Hypertext Perspective	97
6. Personnel and Publications.....	99
6.1. Personnel.....	99
6.2. Major Publications and Presentations.....	99
Appendices	
A. Measures of Vector Similarity	101
A.1. Inner Product.....	101
A.2. Dice Product.....	101
A.3. Cosine Coefficient.....	101
A.4. Jaccard Coefficient	102
B. From Text to HyperText	103
B.1. Introduction	103
B.2. Text Referent Functionalities.....	104
B.2.1. Document Title Field.....	104
B.2.2. Hierarchy Field.....	104
B.2.3. Text Field	104
B.2.4. Built-in Descriptor Zone.....	104
B.3. Construction of an Explicit Hierarchical Structure.....	104
B.4. Graphics Referent Functionalities	105
B.4.1. Sensitive Graphic Area Creation	106
B.4.2. Sensitive Graphic Area Maintenance.....	106
C. Multimedia and Virtual Environments	107
C.1. Multimedia.....	107
C.1.1. History	107
C.1.2. Videodiscs	108
C.1.3. Multimedia Applications.....	108
C.2. Virtual Environments.....	109
C.2.1. The DataGlove.....	110
C.2.2. The Virtual Environment Workstation.....	111
C.2.3. Applications.....	112
D. HyperCard and HyperTalk.....	115
Abbreviations and Acronyms.....	119
References	121

Summary

This technical memorandum (TM) presents the main technical issues of the Computer Integrated Documentation (CID) project. The problem of automation of documents management and maintenance is analyzed both from an artificial intelligence viewpoint and from a human factors viewpoint. Possible technologies for CID are reviewed: conventional approaches to indexing and information retrieval, hypertext, and knowledge-based systems. A particular effort has been made to provide an appropriate representation for contextual knowledge. This representation is used to generate context on hypertext links. Thus, indexing in CID is context-sensitive. The implementation of the current version of CID is described. It includes a hypertext database, a knowledge-based management and maintenance system, and a user interface. This TM also provides a series of theoretical considerations as navigation in hyperspace, acquisition of indexing knowledge, generation and maintenance of a large documentation, and relation to other work.

Acknowledgements

Thanks to Philippa Gander for many useful comments on the work presented in this technical memorandum. Mark Gersh helped in providing a relevant application domain for the techniques developed as well as in the fine-tuning of this report. Nathalie Mathé, Ann Patterson-Hine, Peter Friedland, David Thompson, and Catherine Baudin also provided astute advice towards improving the quality of this technical memorandum.

Chapter 1

Description of the Problem

This chapter introduces the problem of designing a Computer Integrated Documentation (CID) system. Several approaches have been taken in the field of information retrieval and publishing. The current approach is based on intelligent assistant systems (IAS) (section 1.2). We introduce the main problem as being documentation management and maintenance (section 1.3).

1.1. Introduction

The basic task is to build an intelligent problem-driven context-sensitive browsing tool which interacts with and learns from users, and uses *Advanced Interaction Media* (AIM) which include intelligent hypertext, multimedia, and virtual environments. We intend to apply this tool to Space Shuttle and Space Station Freedom (SSF) documentation. As we are designing a generic tool, other applications can be anticipated in such domains as office automation and on-board electronic libraries.

The capacity of an operator to absorb overall Space Shuttle or Space Station information and knowledge, and use it to reach intelligent decisions is stretched not only by the amount and variety of the available data, but also by the complex relationships among different types of information, and the resulting difficulties in interpreting the data.

This research project deals with AIM, i.e., the methods and techniques to generate, analyze, store, retrieve and handle information entities by using specifically designed tools. AIM research differs from, and complements, classical efforts in hypertext or artificial intelligence (AI) in the sense that it integrates both approaches. It tries to attack real-world problems such as technical documentation (in general) and procedures followed in space applications. The main issues and objectives of the Computer Integrated Documentation (CID) project are:

- to provide requirements to build integrated documentation (in the context of the Space Station),
- to examine, better understand, and improve information and knowledge retrieval during operations (in the context of the Space Shuttle).

In the current research effort, we are tackling the difficult problem of design and use of electronic extensions to short- and long-term memory during operations.

1. Requirements for short-term memory problems are handled by operations procedures. Operations procedures are ready-to-use sequences (or simple algorithms) of actions. They are made of shallow knowledge necessary for carrying out well-

understood tasks. They are used during operations as safeguard guidelines and extensions of the short-term memory of operators. Corresponding human information processing involves domain-specific (expert skills) methods.

2. Long-term memory problems happen when operations procedures (i.e., shallow knowledge) are not sufficient. This occurs after unexpected situations where deeper knowledge is necessary to solve non-formalized problems (i.e., problems that do not have procedures as recovery solutions). In these cases, operators have to be able to access deep information and knowledge (a solution would be to provide an extension of the human long-term memory with some appropriate problem solving mechanism and strategies). Corresponding human information processing involves weak methods (classical AI-type problem solving).

Two space applications have been selected to demonstrate these concepts.

1. The exploration of SSF Program Requirement Document as an electronic extension of the long-term memory for the Space Station. CID would allow operations people as well as designers to explore and find out quickly why a piece of equipment has been designed the way it is (design rationale retrieval and use). The analysis on this application should provide guidelines for future documentation systems design. A system is currently under development.
2. Procedures in the mission control room at JSC are being examined and will lead to a generic application of procedure following. The problem is to understand the various mechanisms that are used by flight controllers during flight operations to retrieve and use procedures. A system will be implemented to test the extracted concepts.

1.2. Intelligent Assistant Systems Background

Information retrieval is generally handled using keywords and binary equations of keywords. It happens that this technique is not satisfactory in situations where context (or time) is a critical issue. Keywords are generally built on a context-free basis. Keywords builders even try to decontextualize keywords. In contrast, our approach tries to restore context in keywords.

1.2.1. Motivations for Designing Intelligent Assistant Systems

Progress in aerospace technology stresses the need for higher performance, more reliable and safer systems. The common factor is that the human monitors, controls, diagnoses and maintains systems which are evolving even when the operator does not act on them. This is a process control situation. Documentation is designed and developed for helping operators in such situations.

A major problem of current automation is that the human operator can be either underloaded in normal situations or overloaded in abnormal situations. Furthermore, documentation is often needed when people do not have time to consult it, i.e., when they are overloaded. Moreover, as automation increases, there will be fewer operators for more sophisticated tasks on very complex systems. Thus, it is predictable that operators will not be able to respond in a reasonable period of time in abnormal situations or in very demanding normal situations. The efficiency and quality of their responses will be dependent on their skills in understanding and handling operational situations.

Implementation issues on intelligent assistant systems (IAS) have been already developed (Boy, 1991).

1.2.2. Human Operator Modeling

The emerging model is quite different from "conventional" models of problem solving used in AI. Rasmussen (1986) defines human information processing as a process including a hierarchy of three levels: the skill-based behavior which is a set of well-integrated automatic processes¹ (kinds of stimulus-response processes), the rule-based behavior which corresponds to actual expert systems (IF-THEN rules) and the knowledge-based behavior which corresponds to high level situation identification, decision making, and planning.

When a skilled human operator performs routine tasks, he does not use the rule-based or knowledge-based level but the skill-based level. Human problem solving in routine tasks is guided by "patterns" learned from training and past experience. The operator's real expertise is not to reason analytically while he is under situational pressure. He is accommodating pre-learned procedures in real-time. In some rare cases, such local accommodations cannot work and lead to failure because the operator does not have a ready-to-use procedure. Even in this case, driven by a high workload, the operator will choose general heuristics which he accommodates as specific situational patterns.

This goal-oriented strategy, based on local accommodation of patterns, is very efficient and has been observed in real world operations (Boy et al., 1983, 1985, 1986, 1987, 1988). To date, three projects carried out both at CERT (French Aerospace Administration Research Center in Toulouse) and at NASA-Ames have improved understanding of this model: the MESSAGE project developed for commercial aircraft certification, the HORSES project developed for studying human-machine cooperation in fault diagnosis, and the SAOTS project developed with CNES (French Space Agency) for studying operator assistance in space telerobotics.

A computer model is now available (see the next section on SRAR). It has to be tested and improved on other applications.

1.2.3. Situation Recognition Analytical Reasoning Model

A problem is characterized by a problem statement and a problem solving process leading to a solution. Everybody knows that a well stated problem is already half solved. Moreover, when a beginner starts to learn a particular domain, he starts learning analytical knowledge which he will improve incrementally simply by augmentating his knowledge base and also by transferring various entities of his initial analytical sub-optimal knowledge base towards a situational knowledge² better suited for routine use (more expert).

The SRAR model (Situation Recognition and Analytical Reasoning) provides a formal framework for representing situational (problem statements or situational patterns) and

¹ Automatic processes are taken in the sense of Schneider & Shiffrin (1977).

² The quality of communication between two individuals relies on reciprocal understanding of each other's internal model of thinking. For instance, a discussion between experts of the same domain is carried out in an operative language (Falzon, 1986) which is "very situational". In this case, experts have almost identical knowledge of the subject they are talking about, i.e., their internal models are nearly the same. Conversely, when a professor teaches, his internal model is very different from those of his students'. In particular, a professor has to "decompile" his own situational knowledge to make it understandable by novices. He will be said to be using an "analytical explanation" to make himself understood. This distinction between analytical and situational is not new. In his critique of Artificial Intelligence, Hubert Dreyfus (1979) claims that there is no expertise without situational knowledge. He claims also that it is very difficult to elicit and represent such knowledge in order to use it in computer programs.

analytical (problem solving resources) knowledge. It provides a good framework for representing processes of learning by specialization/structuring. This model is used also for designing an incremental knowledge acquisition tool. At the beginning of the knowledge acquisition process, intelligent systems³ have few small situational patterns (they are inexperienced) and broad analytical knowledge which allows them to solve several problems in the given domain. As the knowledge acquisition process goes on, situational patterns (problem statements) become more complex and numerous, and at the same time the analytical knowledge becomes more structured (Figure 1.1).

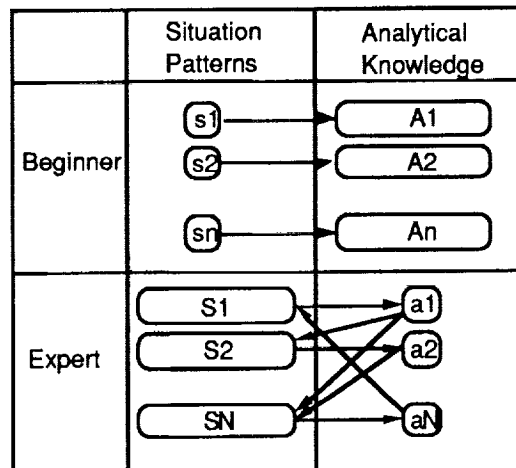


Figure 1.1. The SRAR model.

When an IAS is implemented using the SRAR model, it can be used as a student with the operator being the teacher. It has a very analytical knowledge base at the beginning of its design and it will acquire more situational knowledge with practice. Analytical knowledge may come from first principles or from various episodes or scenarios. For instance, analytical knowledge is included in most manuals or books⁴. In the former case, the problem statement pattern matching will be done by instantiation and the resulting reasoning will be deductive. In the later case, it will be done by analogy. Both of these cases have to be taken into account for building an IAS.

1.2.4. Knowledge Representation Issues

The SRAR model requires an appropriate knowledge representation (KR) which is tractable to both computers and users. The KR currently in use was designed as a representation framework for operation manuals, procedures checklists, user guides and other on line tools useful for controlling complex dynamic systems. The basic entity is a "block" which includes five characteristics: a goal, actions, initial conditions, abnormal conditions and contextual conditions. This knowledge representation is briefly described in this section and developed in section 3.1.

³ This has been observed on humans involved in problem solving tasks, in particular in the HORSES experiment (Boy, 1987). However, we claim that such observations can improve intelligent computer systems also.

⁴ It is generally very difficult to express situational knowledge using written support only. People mix text, drawings, and even gestures to express situational ideas. The main claim of this research is that, because they are good at simulation, computers can provide this kind of ability.

Blocks are organized by context. Several blocks are said to be in the same context when they can be connected to the same overall goal. A situation pattern is a problem statement which, if it matches the current perceived situation, will activate a context of blocks. Blocks are then explored and executed to solve the problem stated by the situation pattern. The execution of a block is started whenever its goal is evoked and its initial conditions are satisfied. The execution of a block consists of performing its actions and controlling the non-satisfaction of the corresponding abnormal conditions. Constraints can be of two types: weak constraints which, if they are not satisfied, will cause an exit from the current block towards another possible block in the same context, and strong constraints which, if they are not satisfied, will cause an exit of the current block towards another possible context of blocks.

Blocks have been designed to capture both analytical and situational aspects of knowledge. Situation patterns are represented by a set of conditions to be satisfied. These conditions can be bounded together by logical or mathematical operators. At the beginning of the knowledge acquisition process, situation patterns are very small, i.e., they include a few very simple conditions. As experimentation continues on the IAS, parts of some blocks can be transferred into situation patterns (Boy & Delail, 1988). For instance, there are actions, like data acquisition, which are performed during the analytical reasoning process, and which could be part of the information to be checked in the first place, i.e., as part of the situation pattern initializing the analytical reasoning. This transformation caused by experience could be called situational learning.

1.2.5. Associated Human Factors Problems

We do not, at the present time, have a rational, predictive methodology for system design by which the AI subsystem developer can integrate human factors principles with other system design principles. We would need to understand better how people recognize patterns, assimilate and integrate information, add their own previous knowledge and value structure, work together, and come up with intelligent, appropriate decisions under difficult circumstances. We would need to understand how actions are initiated, evaluated, and reformulated in the real world. We do not fully understand the constraints that must be placed on the design of the non-human subsystem because of human motivation, action, and experience.

Only recently, the human factors research community started to improve (particularly with regard to human cognition) the consolidation of their empirical data into design methods and principles with which to guide the design process for IASs. Empirical emphasis tends to have been placed upon isolating the underlying properties of individual processes rather than upon considering the human as a single component of some larger, complex, cognitive system. Despite the lack of precise models, a careful analysis of human behavior must play a major role in the design of future human-machine systems. Performance (in particular cognitive performance) assessment is a critical issue that must be at the forefront of this collaborative effort. Since we do not have design guidelines, one of the principal problems that confronts system developers is how to assess the quality of a system composed of artificial intelligence (AI) and Human Intelligence (HI) components, and how to test the system for operation in regions of problem spaces which are beyond the design basis. A method of simulation is needed to explore potential problems of the total system operating in the appropriate scenarios at an early design stage.

In the view of the many uncertainties described above, it seems obvious that a successful design, operation, and evaluation of complex dynamic human-machine systems demands collaboration between the AI and HI communities from the earliest stages of conceptual design. The purpose of the proposed activity is to implement a collaborative AI-HI approach with respect to particular applications.

There are two possible ways to evaluate an engineered system from design to operations.

1. One may use already established human performance models, simulate human-machine interactions, and deduce indices on potential human workload and performance. Such approach has been already used in France using the MESSAGE system for the evaluation of pilots workload and performance in commercial aircraft cockpits (Boy, 1983). The Man-Machine Integration Design and Analysis System (MIDAS) is an existing computer-aided design workstation incorporating a human-performance model with (at present) limited cognitive ability together with a rapid prototyping capability for changing the human environment and mission. It is a human factors engineering tool which assists design engineers in the conceptual phase of crewstation development and helps anticipate training requirements. It provides designers with interactive, analytic, and graphical tools which permit visualization for human engineering principles. Its human-performance model is a broadly based framework upon which various partial models of the cognitive system may be integrated. The argument in support of using human-performance models is straightforward. It is simply not feasible to perform formal human-subject experiments for identifying or resolving all the issues that are likely to arise within the context of a particular system design sufficiently early in the design cycle. Moreover, it has become widely accepted that predictive needs for design purposes are best served by approximate models (Elkind, Card, Horberg & Huey, 1989; McMillan et al., 1989). Analytical methods utilizing human-performance models synthesize disparate results into a unifying theory (normative model) and allow a basis for prediction in new situations. Theories provide bases for design decisions in the absence of specific data.
2. Another approach is *closed-loop rapid prototyping with potential end-user feedback*. This technique is very difficult and often impossible to carry out when end-users are either very busy, unavailable, or not identifiable in advance. In space applications, end-users are generally well identifiable in advance. They are astronauts, ground flight controllers, designers, engineers, etc. What is not well identifiable is the situations in which these people are going to evolve during future missions. Simulation experiments are necessary to train these people.

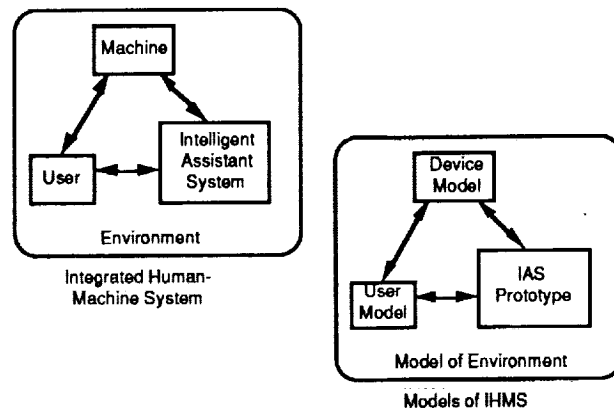


Figure 1.2. The structure of an IHMS, and the corresponding models.

Closed-loop rapid prototyping with potential end-user feedback methods allow both human operator training and incremental evaluation of IASs necessary for safety and success of operations. In this approach, there is no difference between prototype and

end-product. The developed system evolves with both feedback from users and implementation of new technological solutions (coming from the understanding of human-machine limitations and advantages). Closed-loop rapid prototyping with potential end-user feedback methods necessitate cognitive modeling. An approach to the construction of integrated human-machine systems (IHMS) has been given in (Boy & Gruber, 1990). Figure 1.2 illustrates the structure of an IHMS, and the corresponding models. In the present work, the machine is the documentation (i.e., the hypertext), and the intelligent assistant system (IAS) is the context-sensitive indexing and information retrieval mechanism.

1.3. Documents Management and Maintenance

1.3.1. Introduction

The complexity of modern engineered systems stresses the need for "good" technical and operational documentation systems. At present, technical and operational manuals are designed and developed⁵ to satisfy four goals:

- storing systems architecture and internal functions,
- storing performance and limitations (constraints) of these systems,
- reuse of such information for operations and maintenance,
- reuse of such information as a basis for designing new systems.

Previous attempts at computerizing documentation for aerospace systems have been restricted in scope or difficult to use. We have previously developed a knowledge-based system called HORSES designed to assist astronauts in the use of electronic documentation to aid fault diagnosis on the Orbital Refueling System (ORS) of the Space Shuttle (Boy, 1986, 1987). This system was connected to, and tested on, an ORS simulator, and has stimulated the new research directions underlying the present proposal. In particular, advances are needed in: techniques for acquiring domain knowledge and appropriate retrieval strategies, knowledge representation, multimedia systems for interconnecting documents, and user interaction.

In this section, we analyze the problems of increasing volume of technical documentation, availability of documentation experts, disparity of knowledge sources, and on-board documentation.

Increasing volume of technical documentation

Ventura (1988) analyzed the documentation problem in a military perspective. Current fighter aircraft need 300,000 to 500,000 pages of documentation, and this quantity is impossible to deal with in a paper format. The same problem⁶ can be generalized to other industries such as civil aeronautics, nuclear energy, or chemical industries. Even if such documentation can be generated on paper, it is generally not well used if not used at all.

⁵ Note that developers are generally not the end-users. Most of the problems that end-users encounter is usually due to this.

⁶ Rep. Robert A. Roe, who chairs the House Committee on Science, Space and Technology (Asker, 1990), has stated: "In this era of information overload, the bigger question is how do we make judgements about which information is necessary to preserve and protect and which is expendable... This question grows in significance as we enter the space station era, where each day will bring forth enough data to fill the Library of Congress." Big organizations currently develop huge documentation bases which are difficult to trace and search. The problem of selection of the information to keep is solved empirically, case by case.

When Documentation Experts are needed, they may not be expert anymore !

Documentation is very labor-intensive, requiring a great deal of expertise and development time. However, the main problem is that expert developers may no longer be available by the time a very large documentation project is nearing completion. They may have been assigned to a different project or changed employment. In the life cycle of a system, its technical documentation is more useful during operations than during design. Furthermore, it happens that most documentation systems (generally paper-based) are not entirely self-explanatory. Operators generally need help from documentation developers. At this point, developers may not be actively involved in the topic anymore. Thus, they have to carry out a tremendous amount of problem solving activity to understand and retrieve the information needed.

Disparity of knowledge Sources

With present documentation systems, it is difficult to integrate different sources of knowledge in a problem solving task. This integration can be done during the problem solving process (as is done now), i.e., human operators have to construct on-line dynamic bindings between various available sources of knowledge. However, integration can be facilitated if the knowledge is prepared in an appropriate form and if the connections between the various chunks of knowledge are already programmed. As it is very difficult to design an appropriate connection network of knowledge, knowledge transformation will be necessary from an analytical knowledge base to a more situational one. In the current Space Station Freedom documentation (SSFD), integration of various sources of knowledge is a key issue. SSFD specialists have to hierarchize, merge, and integrate documents written by various people.

On-Board Documentation

It is well known that technical paper documentation for an aircraft, for instance, weighs approximatively the same as the aircraft itself. Obviously, such documentation cannot be available on-board. Often, however, all this technical information is not necessary. In very specific cases which generally lead to incidents or accidents, "good" and complete technical information may be very useful for avoiding a catastrophe, e.g., the Chernobyl accident. Thus, computer documentation must be designed to be useful for the operator. At present, operation manuals have been computerized and are available on cathode raytube displays on-board Airbus A320 and Boeing 757 and 767. They include only an operational shallow knowledge sufficient for most operations. However, they do not provide assistance for lower levels of detail, which could be very useful in complex unexpected situations. In such situations, on the Space Station, it will be necessary to get integrated answers inferred from lower levels of information. Hence, shallow knowledge will not be sufficient. An integrated documentation system would be a very important tool for Space Station Freedom because it will decrease the required time to access relevant information and solve problems that are not predictable in advance.

1.3.2. Technical Documentation from Design to Operations

There are two major questions: how can designers benefit from the end-users experience, and how can technical information be represented in order for it to be useful to end-users? Operations people spend enormous amounts of time trying to understand how complex machines work (or why they incidentally do not work as they should). Unfortunately, it is often the case that these same people do not understand the documentation itself or how to

use it. This problem is very difficult to handle in everyday life because it is difficult to write suitable documentation and because it is very difficult to search a huge documentation base without appropriate indexing. Both designers and operations people have to modify the documentation: the former because new directives impose modifications of the design and consequently its documentation, and the latter because new situations force modifications of operations procedures. This aspect leads to a lack of supportability that has been observed during the last decade⁷.

Technical documentation is produced incrementally, no matter if it is generated top-down or bottom-up. People try to incrementally tailor the documentation to their needs or the needs of classes of users. For instance, in modern programming MacLean et al. (1990) noticed that *tailoring can be seen as a process of user evolving the system gradually along with their own changing skills and requirements. So they may have a button of their own, or one provided by a colleague, which does almost what they want, "except for..."*. One of the main expectations from this project is actually to design a user-tailorable documentation system. The corresponding incremental construction mechanism is illustrated in Figure 1.3.

⁷ Lowry and Feaster (1987) have emphasized the life cycle cost (LCC) of a system. They have divided it into four phases: (1) the *mission definition* phase that involves the conceptualization of the system, i.e., definition of the problem to be solved and consideration of initial architectures; (2) the *design* phase including the design itself and the development and test of the prototype; (3) the *production* phase that entails the manufacturing of the product; (4) the *operations* phase which involves training, actual use of the system, maintenance, repairs, etc. The life cycle costs for a military or commercial system are provided in Table 1.1. The Fiscal Year 1985 Congressional Budget Report gives a Space Shuttle LCC distribution reported in Table 1.2. These numbers clearly indicate the importance of operations in the life cycle cost of a system. The main problem is that performance is currently almost the only major criterion taken into account during design, with little or no emphasis on supportability. Lowry and Feaster have analyzed the 1986 Challenger accident. Among the essential issues they identified were accessibility, design criteria, integration, maintainability, management, procedure, reliability, design requirements, standards, training, and certification. All these issues show the current lack of *supportability*. Instead of designing only for performance, can we design for supportability? CID might be a good solution for helping to solve this difficult problem.

LCC Phase	LCC% Cost	
1. Definition	< 1	%
2. Design	< 10	%
3. Production	30	%
4. Operations	60	%

Table 1.1. Department Of Defense LCC Distribution (Lowenstein & Winter, 1986)

LCC Phase	LCC% Cost	
1. Definition	< 1	%
2. Design	6	%
3. Production	8	%
4. Operations	86	%

Table 1.2. Space Shuttle LCC Distribution (Fiscal Year 1985 Congressional Budget, 1985)

Generally, documentation is built and used to achieve a given goal (that may be subdivided into several subgoals). It is modified by assessing the results of its use in several contexts (see section 1.3.4). Modifications may affect its indexing mechanism, its infrastructure, and/or its content.

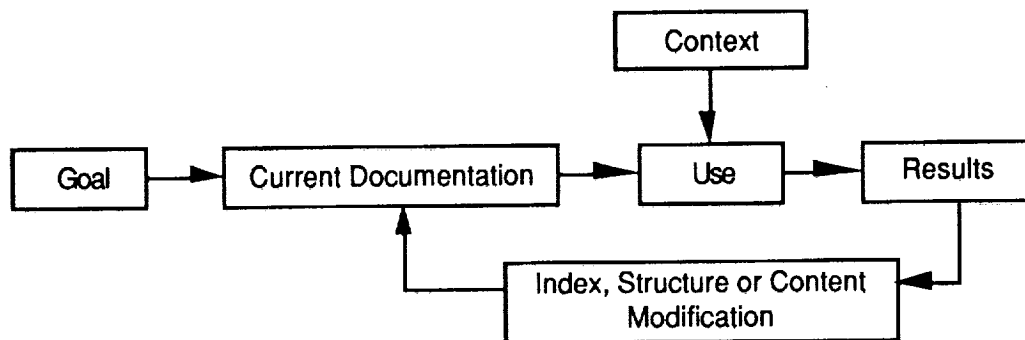


Figure 1.3. Documentation construction mechanism.

1.3.3. Navigation Problems in Documentation

Users can navigate into the documentation either using the table of contents, the index, or their own indexing mechanisms and strategies. They generally build their own cognitive map of the documentation. Studies have been carried out on cognitive navigation in 3-dimensional space (Oatley, 1977). Navigation in n -dimensional spaces, or hyperspaces, is recent and deserves more investigations. Our current research focus is on hyperspace navigation.

The table of contents is generally a good road-map that allows hierarchical navigation in a document. The index is a rough "what to do next" precompiled keyword search mechanism. Keywords (also called descriptors) are generally grouped in alphabetical order in the index. This is generally a good tool to search concepts. As a metaphor, when people need to find their way in an unknown region, they use a map to locate themselves in the region and to construct a strategy to go from one place to another. Sometimes the map is not sufficient and they need to ask a knowledgeable person "what to do next".

Indexing is a key process in the navigation problem. Indexing is based on experience. The corresponding knowledge is usually acquired incrementally from a sub-optimal knowledge base. Generally, indexes (at the end of a book, for instance) are context-free. However, the more an index is used the more it becomes context-sensitive. People tend to write annotations, put meaningful colors to highlight particular features, underscore sentences, etc., in order to contextualize the initial indexical knowledge. In other words, people build their own procedures to retrieve information faster and in a more appropriate and accurate manner.

1.3.4. Context

Why does information retrieval often fail? First, information retrieval is generally done using keywords linked by logical operators. These keywords are designed and assigned in a given context that the designers try to make as general as possible. However, the designers (of these keywords) cannot predict all contextual situations in which the potential users will use them. The notion of *context* is crucial in navigation problems. The notion of

context is close to the notion of point of view (or extension) developed in belief revision theories (Doyle, 1979; de Kleer, 1986).

Context could be defined as a set of contextual conditions (also called situation pattern) that hold in a given situation. Documentation designers cannot anticipate what end-users will need and use in the documentation they are developing. They do not know how they will enter into the documentation. Use of documentation is very context-sensitive. For instance, let us assume that you want to retrieve some very specific information on the air conditioning in the main cabin of the Space Station. The first thing you may try is to select the index "air conditioning" in your documentation and browse it with this index. If you can specify the context of your retrieval, e.g., "you are a designer and are concerned by the connection of the air conditioning system, and have very little information about the electrical circuitry in the cabin", then you will specify a better search in the documentation. The search will not be the same if you mention that "you are an astronaut in the Space Station and are freezing". Contextual conditions are acquired by experience. It takes hours (sometimes years) to attach context to problems. In other words, stating a problem requires the good contextual conditions, if one wants to solve this problem more easily.

It is very difficult to elicit such contextual conditions from experts. This is due to the fact that such knowledge is very compiled. However, if we consider the reasonable assumption that contextual knowledge is acquired incrementally, then incremental knowledge acquisition techniques could be useful for on-line elicitation of context. Indeed, it is difficult for expert users to attach the right situation to any information retrieval strategy, simply because they do not remember well what they would do in any given situation. It is however, very easy to ask them to describe the relevance of retrieved information just after the fact (i.e., on-line elicitation). Obviously, the question is how to ask such additional information from users without overloading or "annoying" them. One partial answer is certainly to reduce the amount of interaction users will have to perform to accomplish this additional task. "Not too long" positive feedback from the system is also an important factor. In other words, context acquisition by the system should be rapidly transparent to users in a short period of time.

1.3.5. Knowledge-Based Indexing

Let's take a preliminary example that will help to understand knowledge-based indexing. When I first started using the text processing system used to write this report, I had an interesting experience trying to display the footnote of a page without adding more information in it. I tried to use the paper documentation provided with the software, but could not find any satisfactory answer. I finally asked someone in our laboratory who was an expert of this text processing system and in a few seconds my problem was solved. What happened is that I used his indexing knowledge to relate my request to the right information necessary to solve my problem. It would be a great help for users if they could have the best expert interacting with them to help them retrieve appropriate information upon request.

The basic model we will adopt is based on the separation between an information base or data base, called the documentation, and a knowledge-base that includes the best knowledge we have to index the documentation according to context.

Useful knowledge to index documentation includes types of users, types of tasks that induce requests, temporal information that is likely to focus retrieval, and other dimensions that would help narrow the search strategies. Eventually, contextual conditions could be triggered automatically if appropriate sensors are available. In some suitable cases, such automated configuration could provide appropriate and timely information to users needing it without any actions required from them.

THE UNIVERSITY OF CHICAGO
DIVISION OF THE PHYSICAL SCIENCES
DEPARTMENT OF CHEMISTRY
5708 S. UNIVERSITY AVE.
CHICAGO, ILL. 60637

Chapter 2

Possible Technologies

In this chapter, we will attack the problems of indexing, information retrieval, current organizational software supports, and knowledge-based system technology available for improving documentation management and maintenance.

2.1. Conventional Approaches to Indexing and Information Retrieval

2.1.1. Definitions

2.1.1.1. Indexing

The process of constructing document surrogates by assigning identifiers to document items is known as *indexing* and is done from text or image analyses. Currently, there are techniques that allow such analyses to be performed automatically but are only used when pure text is available. In general, indexing is done by humans, especially when pictures and graphics are included in the documentation without text describing them.

We also dissociate an *objective* from a *subjective* document identifier (Salton, 1989). A document item, e.g. a paragraph, a chapter, a volume, a library, can be indexed by an objective identifier that can be its title, its author(s) and its publication date, its number, etc. Generally, tables of contents provide objective document identifiers. Conversely, subjective identifiers, e.g. keywords, icons, etc., can also be attached to documents items. Subjective identifiers are generally good descriptions of the document piece they identify. However, the main problem with subjective identifiers is that they generally lead to several document items.

2.1.1.2. Descriptors and Referents

We call a *descriptor* any piece of text (word, sentence, or paragraph) or image (marked area or label on part of an image) that describes objectively or subjectively any other piece of documentation. Descriptors can be single-term or multiple-terms. We call a *referent* any piece of documentation (word, line of text, paragraph, picture, moving video or animated sequence, program, volume, library) that is described by at least one descriptor. Referents are sometimes called targets (Martin, 1990). Referents are always characterized by an objective identifier, but they also can have subjective identifiers. Let r be a referent, $\{d_1,$

d_2, \dots, d_n) a set of descriptors each of them independently describing r . There must be at least one d_i that objectively describe r . Generally, the other descriptors (other than d_i) are added because they are more convenient for users, even if they do not objectively describe r .

Descriptors are always included in the content of a referent. This referent can be either the table of contents, the index, or any part of the documentation. Tables of contents or indexes are usually found more convenient because of their well known structure (hierarchical and alphabetical). We call a *consistent set of referents* any set of referents that is described by the same descriptor and can be dissociated from other sets in the current state of the indexing system.

2.1.1.3. Recall and Precision

Two main parameters are important in information retrieval, *indexing exhaustivity* and *descriptor specificity*. Indexing exhaustivity reflects the degree to which all aspects of the subject matter of a document item are actually recognized in the indexing product. When indexing is exhaustive, a large number of descriptors are often assigned, and even minor aspects of the subject area are reflected by corresponding descriptor assignments.

Term specificity refers to the degree of breadth or narrowness of the descriptors. When broad descriptors are used for indexing, many useful information items are likely to be retrieved for users, together with a substantial proportion of useless materials. Normally, broad descriptors cannot distinguish relevant from nonrelevant document items. Narrow descriptors, on the other hand, retrieve relatively fewer items, but most of the retrieved materials (referents) are likely to be helpful to users.

People working in information retrieval have developed formulas that measure indexing exhaustivity and descriptor specificity. *Recall* (R) is the proportion of relevant material that is retrieved:

$$R = \frac{\text{Number of relevant referents retrieved}}{\text{Total number of relevant referents}}$$

Precision (P) is the proportion of retrieved material that is relevant:

$$P = \frac{\text{Number of relevant referents retrieved}}{\text{Total number of referents retrieved}}$$

When a choice must be made between extreme descriptor specificity and extreme descriptor breadth, the former is generally preferable because the output produced by the high-recall, low-precision alternative tends to burden users with unmanageably large piles of retrieved materials. In contrast, high precision searches retrieve fewer items that are more easily examined (Salton, 1989).

In order to satisfy these recommendations, we currently choose to add context to descriptors for improving both precision and recall. At the extreme limit, from a descriptor in a well defined context, we should retrieve only one consistent set of referents.

2.1.2. Single-Term Descriptor Indexing

2.1.2.1. Frequency-Based Methods

If we consider descriptors as single-terms, systematic full-text search can be performed to extract all the words from a given free-text document. Fast procedures, like TEX, are

already developed for accomplishing this first step (Zimmerman, 1988). TEX provides a list of all the words attached with their number of occurrence, and allows users to visualize each word in its context⁸ in the corresponding document. Such facility allows users to rapidly separate *domain descriptors* from *non-domain descriptors*. At this stage, words morphologically related, e.g., plurals, tenses, etc., are not processed and kept as is. Both domain descriptors and non-domain descriptors are kept in a long-term memory which will be used for further extractions of single-term descriptors. Among the non-domain descriptors, there are function words, e.g. "and", "of", "or", "but", etc., that have approximatively the same frequency of occurrence in all document items of a library (Salton, 1989).

Luhn (1957) proposed an indexing method based on the number of occurrence of single-term descriptors. The corresponding algorithm is presented in Figure 2.1.

-
1. Eliminate common function words from the document item by consulting a special dictionary (also called stop list) containing a list of high-frequency function words.
 2. Compute the single-term descriptor frequency $f(d_j, r_i)$ for all remaining descriptors d_j in each referent r_i , specifying the number of occurrences of d_j in r_i .
 3. Choose a threshold frequency T , and assign to each referent r_i all the descriptors d_j for which $f(d_j, r_i) > T$.
-

Figure 2.1. Indexing Algorithm of Yankelovich, Meyrowitz and van Dam

Note 1. The level of granularity of a referent is variable. A referent can be a single-term descriptor. In this case, the indexing method described above leads to the construction of a thesaurus where the links between the descriptors are frequency-based. A referent can be a section in a document or a document. If there are several libraries or collections, they can be indexed as referents also.

Note 2. The single-term descriptor frequency can be seen as a good approximation of the recall variable (see section 2.1.1.3). Indeed, if a single-term descriptor is repeated several times in a referent, then there is a reasonable chance to expect that this referent deals with such a single-term descriptor. Unfortunately, frequency does not solve the problem of precision. We will have several referents having the same single-term descriptor in their list of descriptors. This is again a good reason to put context around descriptors.

Note 3. Precision is better served by descriptors that occur rarely in referents. If $n(d_j|D)$ is the number of referents in a documentation D in which a descriptor d_j occurs, i.e. given a descriptor d_j it has $n(d_j|D)$ possible referents in a documentation, Spark Jones (1972) found that a typical referent discriminator is given by $\log N/n(d_j|D)$, where N is the number of referents in the documentation. A typical combined descriptor importance indicator w_{ij} of this type is the product of the descriptor frequency by the inverse referent frequency:

$$w_{ij} = f(d_j, r_i) \cdot \log \frac{N}{n(d_j|D)}$$

⁸ Here, context means the text surrounding the extracted word.

Using w_{ij} the above algorithm can be significantly improved by the indexing algorithm presented in Figure 2.2.

-
1. Eliminate common function words from the document item by consulting a special dictionary (also called stop list) containing a list of high-frequency function words.
 2. Compute the combined descriptor importance indicator w_{ij} for all remaining descriptors d_j in each referent r_i .
 3. Choose a threshold frequency T , and assign to each referent r_i all the descriptors d_j for which $w_{ij} > T$.
-

Figure 2.2. Descriptor/Referent Frequency-based Indexing Algorithm

2.1.2.2. Descriptor-Discrimination Value

A good way to discriminate between referents is to measure the distance between their respective sets of descriptors. The more descriptors are shared by two referents the more these referents will be perceived as similar. Similarity between r_p and r_q is expressed by the function $\text{sim}(r_p, r_q)$ as a function of the number of descriptors shared by both referents. Implicating that the more high-frequency descriptors are kept, the more likely referents risk being similar. ∂v_j is the descriptor-discrimination value of the descriptor d_j and is expressed as the difference between space densities before (Q) and after (Q_j) the assignment of descriptor d_j to the referents of the documentation:

$$\partial v_j = Q - Q_j$$

where:

$$Q = \frac{1}{N(N-1)} \sum_{p=1}^N \sum_{\substack{q=1 \\ p \neq q}}^N \text{sim}(r_p, r_q)$$

∂v_j can be combined with the descriptor frequency to build a new weighting formula that can be used to refine descriptor assignments in executing step 3 of the algorithm in Figure 2.2:

$$w_{ij} = f(d_j, r_i) \cdot \partial v_j.$$

2.1.2.3. Conclusion

There are other methods for single-term descriptor indexing. In particular, when we want to take into account potential important distinctions between descriptor occurrences in relevant and nonrelevant referents. The probabilistic term weighting model (Salton, 1989) makes explicit distinctions between occurrences of descriptors in the relevant and nonrelevant referents of a document. We will not describe this model here. However, the concept of relevance will be taken into account in our knowledge-based indexing mechanism.

The use of single-term descriptor indexing is justified only by operational considerations. The use of single-term descriptors introduces ambiguities by its lack of context, and also because single-term descriptors are either too specific or too broad to be useful in indexing. Again, the concept of context is popping up! Generally, when people are doing indexing manually, they naturally construct descriptors that are compound sets of words, such as noun phrases.

We can use the following matrix to build classes of descriptors that have similar assignment to a set of referents (column processing), or classes of referents that have similar assignment to a set of descriptors (row processing).

	d_1	d_2	...	d_m
r_1	w_{11}	w_{12}	...	w_{1m}
r_2	w_{21}	w_{22}	...	w_{2m}
.				
.				
r_n	w_{n1}	w_{n2}	...	w_{nm}

where w_{ij} is the value of the descriptor d_j in the referent r_i .

An usual method for narrowing descriptors is the generation of phrases (compound descriptors) consisting of sequences of single-term descriptors. Thesauruses, on the other hand, can be used for descriptor broadening by replacing individual narrow descriptors with the thesaurus groups in which the descriptors are included. We will develop both of these methods in the following.

2.1.3. Compound Descriptor Formation

It is obvious that "Christmas tree" or "genealogical tree" is more specific than "tree". More generally, we need to build compound descriptors when single-term descriptors have high frequency to increase discrimination. Nonlinguistic (frequency-based) and linguistic methods are generally needed in concert to build compound descriptors. The former methods provide the necessary single-term descriptors that need to be augmented (by computing their frequency and comparing it to a given threshold). The latter methods provide a way to appropriately combine single-term descriptors together with other words.

If two compound descriptors have the same meaning (semantically identical) but are syntactically different, it is usually better to keep both instances of the same meaning for further pattern matching purposes. In other words, it is generally better to keep several aliases of a descriptor to improve the information retrieval. For instance, "information retrieval" and "retrieval of information" have the same meaning but do not have the same syntax, and such descriptors may appear in one or the other form in a referent that we are looking for.

2.1.4. Thesaurus Generation

If a descriptor is too narrow, i.e. its frequency is very small, then it may be convenient to broaden its scope by attaching to it a set of other descriptors that are more general than the descriptor itself. Let us give some useful rules that can be used to generate a thesaurus. A

thesaurus will be represented as a semantic network where nodes are descriptors and links are relations between these descriptors.

2.1.4.1. Enlarging Pattern-Matching Capabilities

Generally, descriptors have to be fully matched to get successful results. However, it often occurs that appropriate transformation of descriptors may improve pattern matching capabilities, such as removing suffixes recursively from the tail ends of words.

Removing suffixes.

1. Define a dictionary of suffixes, e.g., -ness, -ing, -er, -y, -ic, -ical, etc.
2. Define a set of rules for handling exceptions, e.g.,
 - restore a silent e after suffix removal from certain words to produce "hope" from the original "hoping" rather than "hop";
 - delete certain double consonants such as b, d, g, l, m, n, p, r, s, and t after suffix removal, so as to generate "hop" from "hopping" rather than "hopp";
 - use a final y for an i in form such as "easier", so as to generate "easy" instead of "easi".

Other capabilities include the detection of spelling mistakes or incomplete spelling.

2.1.4.2. Aliases

Aliases are terms defined for the same concept. Generation of aliases is also a good way to enlarge the scope of an initial descriptor. It is usually done manually. For this reason, end-users should have the possibility to generate their own aliases. Aliases can be abbreviations, acronyms, or different names having the same meaning as the original descriptor.

2.1.4.3. Conclusion

Constructing a thesaurus in a given subject area is always demanding. This process is usually performed manually even if automatic thesaurus-construction systems already exist. Those are generally classifiers. Furthermore, the effectiveness of thesauruses they produce is questionable outside the special environment in which they are generated (Salton, 1989). Enlargement of pattern-matching capabilities and aliases are not the only possibilities for constructing a thesaurus, they have been given as examples. More generally speaking, related descriptors will be constructed around a given descriptor with appropriate semantic links.

2.1.5. Some Information Retrieval Models

In this section, we will present four models that are used to retrieve information:

- vector space model,
- automatic document classification,
- probabilistic retrieval model,
- fuzzy set retrieval model.

2.1.5.1. Vector Space Model

2.1.5.1.1. Description of the method

Goal of the method: compute similarity coefficients between queries and referents.

Let W be the descriptor-referent matrix:

	d_1	d_2	...	d_m
r_1	w_{11}	w_{12}	...	w_{1m}
r_2	w_{21}	w_{22}	...	w_{2m}
...				
...				
r_n	w_{n1}	w_{n2}	...	w_{nm}

where $\mathbf{r}=\{r_1, r_2, \dots r_n\}$ are n distinct referents, $\mathbf{d}=\{d_1, d_2, \dots d_m\}$ are m distinct descriptors, w_{ij} represents the value of descriptor d_j in referent r_i . We will adopt the vector notation:

$$\mathbf{r} = W \cdot \mathbf{d}$$

Similarly, let V be the query matrix:

	d_1	d_2	...	d_m
q_1	v_{11}	v_{12}	...	v_{1m}
q_2	v_{21}	v_{22}	...	v_{2m}
...				
...				
q_n	v_{n1}	v_{n2}	...	v_{nm}

where $\mathbf{Q}=\{q_1, q_2, \dots q_n\}$ are n distinct queries, $\mathbf{d}=\{d_1, d_2, \dots d_m\}$ are m distinct descriptors, v_{ij} represents the value of descriptor d_j in query q_i . We will adopt the vector notation:

$$\mathbf{Q} = V \cdot \mathbf{d}$$

Typically, if we assume boolean descriptor membership to referents, we have:

If d_j is in r_i then $w_{ij} = 1$ else $w_{ij} = 0$
 If d_j is in q_i then $v_{ij} = 1$ else $v_{ij} = 0$.

If we assume continuous membership, then values belong to the interval [0,1]. In both cases, if we represent descriptors as vectors in a hyperspace, then a referent can be represented as linear combination of descriptors such that:

$$r_i = \sum_{j=1}^m w_{ij} d_j$$

Given a referent r_s and a query q_p represented in the linear combination form presented above, the referent-query similarity can be computed as:

$$q_p \cdot r_s = \sum_{i,j=1}^m v_{pi} w_{sj} d_i \cdot d_j$$

In practice, it is generally assumed that the descriptors are not correlated, i.e., vectors are orthogonal:

$$\begin{aligned} \text{for } i \neq j, d_i \cdot d_j &= 0 \\ \text{for } i = j, d_i \cdot d_i &= 1. \end{aligned}$$

This assumption leads to following expression of the referent-query similarity:

$$\text{sim}(q_p, r_s) = q_p \cdot r_s = \sum_{i=1}^m v_{pi} w_{si}$$

The same kind of computation can be used to measure the similarity between referents:

$$\text{sim}(r_p, r_s) = r_p \cdot r_s = \sum_{i=1}^m w_{pi} w_{si}$$

2.1.5.1.2. Advantages of the method

There are three main advantages to computing similarity coefficients between queries and referents.

1. Referents can be arranged and displayed to users in decreasing order of corresponding similarity with the query.
2. The most relevant referents can be displayed according to a threshold applied to these similarity coefficients.
3. Referents retrieved early in the search, which are most similar to the query, may help generate improved query formulation using relevance feedback (Salton, 1989).

The measure introduced in section 2.1.5.1.1, i.e., the inner product (or sum of products) is not the only possible measure. There are other normalized measures that are presented in Appendix A.

2.1.5.1.3. Disadvantages of the method

The orthogonality assumption in the basic vector-processing model is the major disadvantage in this method. Indeed, independence between descriptors is not guaranteed. However, descriptor correlations can be given (or assumed) to compute the similarities.

2.1.5.2. *Automatic Referent Classification*

2.1.5.2.1. Description of the method

This type of method focuses on the fact that descriptors of related referents should appear close together. In other words, if referents are characterized by patterns of descriptors, a set of related referents should lead to the same *cluster* of descriptor patterns. Figure 2.3 presents various referents organized by clusters. Each elementary cluster has a centroid. These elementary clusters are classified into superclusters that have a supercentroid. The overall documentation has an hypercentroid that represents the highest level of clustering. The advantage of the referent classification is that the search strategy for clustered referents is equivalent to a tree search such as presented in Figure 2.4.

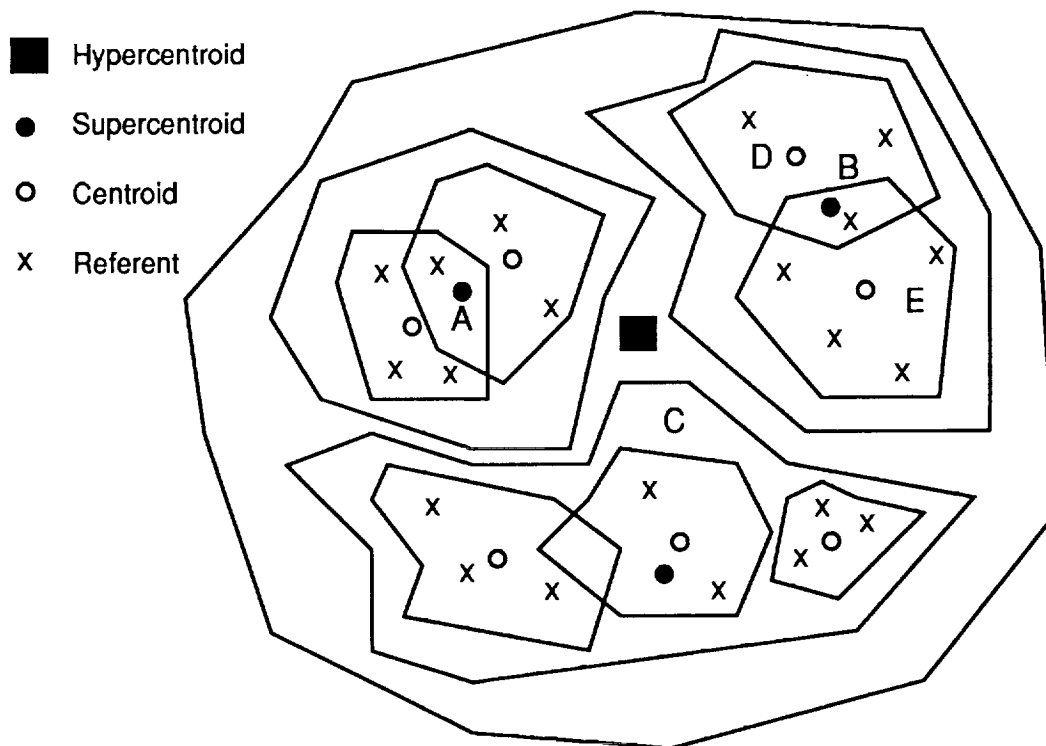


Figure 2.3. Typical clustered referent organization.

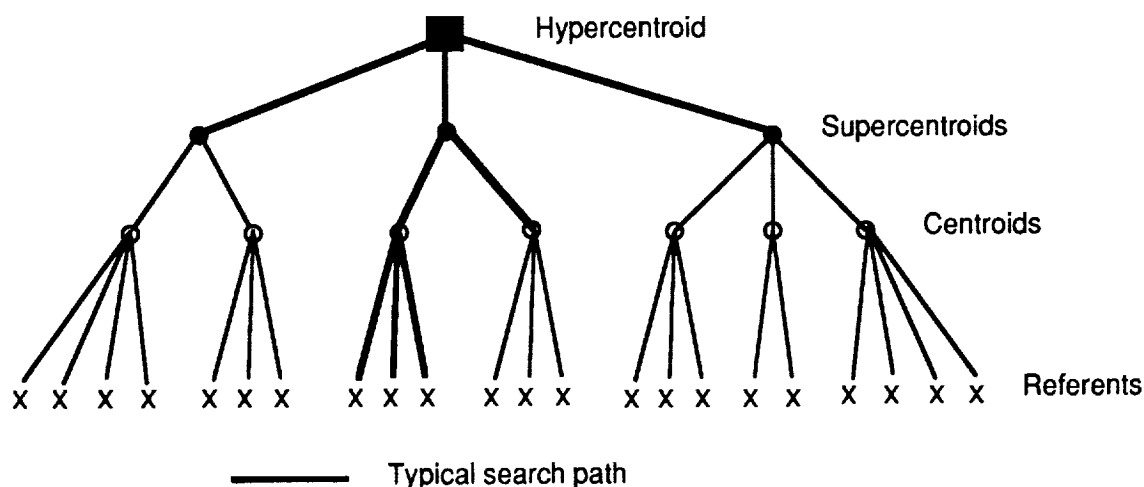


Figure 2.4. Search strategy for clustered referent organization of Figure 2.3.

2.1.5.2.2. Hierarchical cluster generation

The basic algorithm for hierarchical incremental clustering is presented in Figure 2.5.

1. Compute all pairwise referent-referent similarity coefficient [$N(N-1)/2$ coefficients] using the last formula of section 2.1.5.1.1 or an equivalent from Appendix A.
2. Place each of N referents into a class of its own.
3. Form a new cluster by combining the most similar pair of current clusters i and j ; update similarity matrix by deleting the rows and columns corresponding to i and j ; calculate the entries in the row corresponding to the new cluster $i+j$ according to a chosen cluster generation strategy.
4. Repeat step 3 if the number of clusters left is greater than 1.

Figure 2.5. Hierarchical cluster generation algorithm.

Cluster generation strategies.

1. *Single-link clustering*: only the most similar pairs of referents are kept, i.e., for each row p the similarity coefficient for the new cluster are computed as follows:

$$\max_{s = 1, n} \text{sim}(r_p, r_s)$$

2. *Complete-link clustering*: only the least similar pairs of referents are kept, i.e., for each row p the similarity coefficient for the new cluster are computed as follows:

$$\min_{s = 1, n} \text{sim}(r_p, r_s)$$

3. *Group-average clustering*: the similarity coefficients of the new cluster are computed as the average pairs of referents, i.e., for each row p the similarity coefficient for the new cluster are computed as follows:

$$\frac{1}{n} \sum_{s=1}^n \text{sim}(r_p, r_s)$$

Hierarchical cluster generation methods are generally very expensive in calculation time. For n referents this kind of method requires $N^2 \log N^2$ operations. However, they provide a unique set of well-formed clusters for each set of data. Furthermore, the resulting cluster hierarchy is stable, i.e., small changes in input data do not lead to large rearrangements in the cluster structure.

Example of single-link clustering.

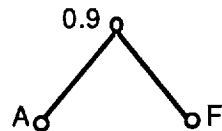
Let $\{A, B, C, D, E, F\}$ be a set of $n=6$ referents. Let the similarity matrix between pairs of referents be the following:

	A	B	C	D	E	F
A	-	0.8	0.5	0.6	0.8	0.9
B	0.8	-	0.4	0.5	0.7	0.8
C	0.5	0.4	-	0.3	0.5	0.2
D	0.6	0.5	0.3	-	0.4	0.1
E	0.8	0.7	0.5	0.4	-	0.3
F	0.9	0.8	0.2	0.1	0.3	-

Step 1:

$$\max_{p=1,6} \max_{s=1,6} \text{sim}(r_p, r_s) = \text{sim}(A, F) = 0.9$$

The similarity pair is AF, and the single-link structure is:



The new similarity matrix is:

	AF	B	C	D	E
AF	-	0.8	0.5	0.6	0.8
B	0.8	-	0.4	0.5	0.7
C	0.5	0.4	-	0.3	0.5
D	0.6	0.5	0.3	-	0.4
E	0.8	0.7	0.5	0.4	-

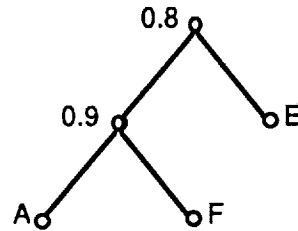
This matrix has been recomputed by using the following formula:

$$\forall r_s \in \{B, C, D, E\}, \text{sim}(AF, r_s) = \max [\text{sim}(A, r_s), \text{sim}(F, r_s)]$$

Step 2:

$$\max_{p=1,5} \max_{s=1,5} \text{sim}(r_p, r_s) = \text{sim}(AF, E) = 0.8$$

The similarity pair is AE, and the single-link structure is:



The new similarity matrix is:

	AEF	B	C	D
AEF	-	0.8	0.5	0.6
B	0.8	-	0.4	0.5
C	0.5	0.4	-	0.3
D	0.6	0.5	0.3	-

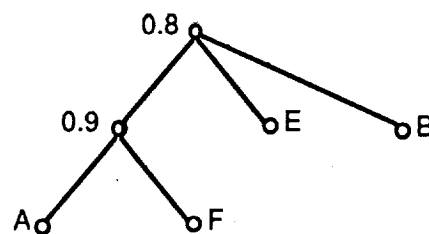
This matrix has been recomputed by using the following formula:

$$\forall r_s \in \{B, C, D\}, \text{sim}(AEF, r_s) = \max [\text{sim}(AF, r_s), \text{sim}(E, r_s)]$$

Step 3:

$$\max_{p=1,4} \max_{s=1,4} \text{sim}(r_p, r_s) = \text{sim}(AEF, B) = 0.8$$

The similarity pair is BF, and the single-link structure is:



The new similarity matrix is:

	ABEF	C	D
ABEF	-	0.5	0.6
C	0.5	-	0.3
D	0.6	0.3	-

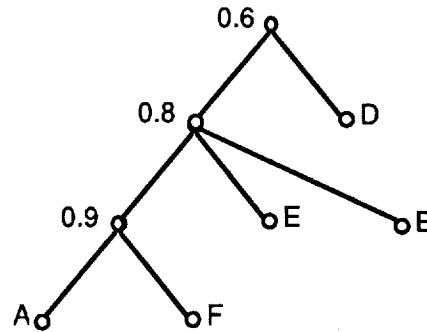
This matrix has been recomputed by using the following formula:

$$\forall r_s \in \{C, D\}, \text{sim}(\text{ABEF}, r_s) = \max [\text{sim}(\text{AEF}, r_s), \text{sim}(\text{B}, r_s)]$$

Step 4:

$$\max_{p=1,3} \max_{s=1,3} \text{sim}(r_p, r_s) = \text{sim}(\text{ABEF}, D) = 0.6$$

The similarity pair is AD, and the single-link structure is:



The new similarity matrix is:

	ABDEF	C
ABDEF	-	0.5
C	0.5	-

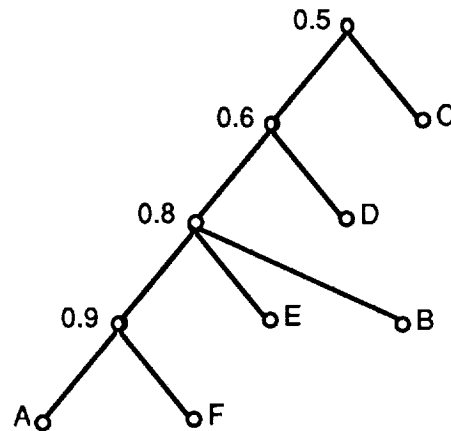
This matrix has been recomputed by using the following formula:

$$\text{sim}(\text{ABDEF}, C) = \max [\text{sim}(\text{ABEF}, C), \text{sim}(D, C)]$$

Step 5:

$$\max_{p=1,2} \max_{s=1,2} \text{sim}(r_p, r_s) = \text{sim}(\text{ABDEF}, C) = 0.5$$

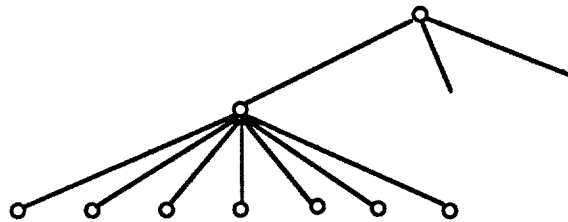
The similarity pair is AC, and the final single-link structure is:



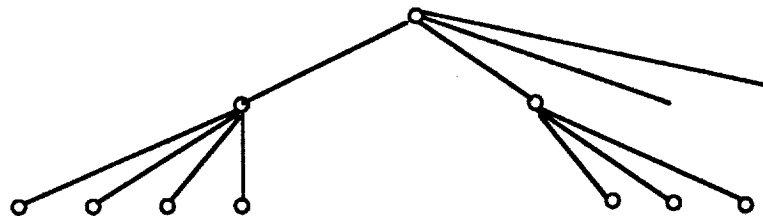
2.1.5.2.3. Heuristic clustering methods

Heuristic clustering methods produce rough cluster arrangements rapidly at relatively little expense. Generally, it is interesting to introduce heuristics to reduce undesirable cluster structures, such as:

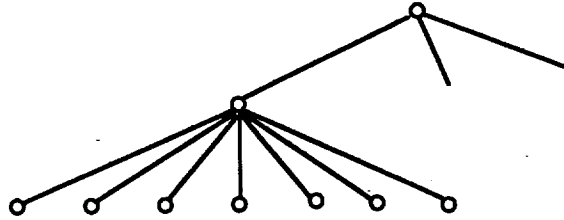
- limiting the number of elements in a cluster by splitting this cluster into several clusters having a suitable size, for instance the following structure:



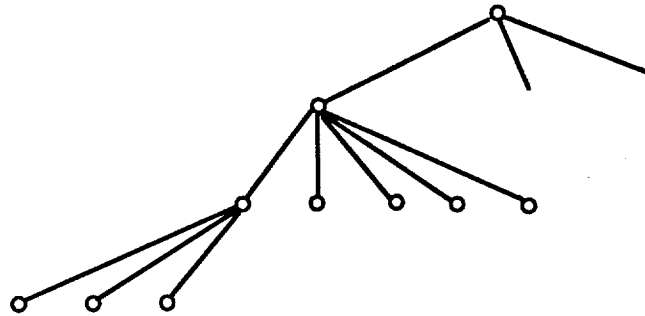
can be rearranged in the following way:



- isolating dense regions of the documentation, i.e., identifying elements in close proximity with other elements (both referents and clusters), and creating a new cluster seed. For instance, in the above example, if the first three referents are defining a dense region, the following transformation can be performed:



to:



2.1.5.2.4. Search in cluster trees

Once a cluster tree has been built, conventional search methods can be used. Generally best first methods are used that take into account similarity coefficients on the tree nodes.

2.1.5.3. *Probabilistic Retrieval Model*

Probabilistic retrieval methods are likely to give good results if they are validated with reasonable sampling. The probabilistic model can include descriptor dependencies and relationships, and major parameters such as the weighting of query descriptors and the form of query-referent similarity are determined by the model itself. The corresponding model is based on the probability of relevance $P(\text{rel})$ of a referent. The probability of nonrelevance is defined as:

$$P(\text{nonrel}) = 1 - P(\text{rel}).$$

a_1 (a_2) is a cost parameter associated with the retrieval of a nonrelevant (relevant) referent. The main idea is to satisfy the corresponding relation:

$$a_2 \cdot P(\text{rel}) \geq a_1 [1 - P(\text{rel})]$$

or:

$$g = \frac{P(\text{rel})}{1 - P(\text{rel})} - \frac{a_1}{a_2} > 0$$

If we assume that the cost parameter a_1 and a_2 are equal, the evaluation of g will be done using other parameters w , i.e.:

$$g(w) + 1 = \frac{P(\text{rel}|w)}{1 - P(\text{rel}|w)} = \frac{P(\text{rel}|w)}{P(\text{nonrel}|w)}$$

Using Bayes' theorem:

$$g(w) + 1 = \frac{P(w|rel)}{P(w|nonrel)} \times \frac{P(rel)}{P(nonrel)}$$

That leads to the logarithmic expression (that must be non negative):

$$\log (g(w) + 1) = \log \frac{P(w|rel)}{P(w|nonrel)} + \log \frac{P(rel)}{P(nonrel)}$$

where $P(rel)$ ($P(nonrel)$) is the a priori probability of relevance (nonrelevance) of any referent. Assuming that the descriptor distributions in the referents of the documentation follow the Poisson binomial distribution, the quantity $P(x|rel)$ can be derived from:

$$P(w|rel) = \prod_{i=1}^m p_i^{w_i} (1-p_i)^{1-w_i} [1+A]$$

where A is a formula given by Bahadur and Lazarsfeld (Yu, Luk & Siu, 1979), $w=(w_1, w_2, \dots, w_m)$ a collection of binary terms, and p_i is the occurrence probability of descriptor d_i in the relevant referents, i.e. $p_i = P(w_i=1|rel)$.

In practice, there is no hope to compute A because there will not be enough data to do it. One assumption would be to assume $A=0$. In this case, the probabilistic model becomes a form of vector space model (see section 2.1.5.1). Another alternative would be to take into account only some of the more important pairwise descriptor correlations⁹, and use the well-known tree-dependence model where all descriptors are assumed to depend on exactly one other descriptor in the descriptor set.

⁹ In that case, it follows:

$$P(w|rel) = \prod_{i=1}^m P(w_{m_i}|w_{m_{j(i)}}) \quad 0 \leq j(i) < i$$

where

$$\begin{aligned} P(w_k|w_{m_0}) &= P(w_k) \\ P(w_i|w_{j(i)}) &= [p_i^{w_i} (1-p_i)^{1-w_i}]^{w_{j(i)}} [q_i^{w_i} (1-q_i)^{1-w_i}]^{w_{j(i)}} \\ p_i &= P(w_i=1|w_{j(i)}=1) \\ \bar{p}_i &= P(w_i=1|w_{j(i)}=0) \end{aligned}$$

After substitutions, the following formula is obtained:

$$\begin{aligned} \log P(w|rel) &= \sum_{i=1}^m (w_i \log p_i + (1-w_i) \log(1-p_i)) \\ &+ \sum_{i=1}^m \left[w_{j(i)} \log \frac{1-p_i}{1-\bar{p}_i} + w_i w_{j(i)} \log \frac{p_i(1-\bar{p}_i)}{\bar{p}_i(1-p_i)} \right] + \text{constants} \end{aligned}$$

2.1.5.4. Fuzzy Set Retrieval Model

2.1.5.4.1. Quantification of Imprecision

The following queries are imprecise :

q_1 = This book was published recently

q_2 = This book was ordered before 1984

Fuzzy sets

The query q_2 represents a simple imprecision concerning the value of the date, i.e. the book is an element of the set of books ordered during the interval [1900, 1984], for example. This is a weighted interval. The query q_1 corresponds to an imprecise predicate (Recently). If the predicate Recently is defined on a set of dates, it is impossible to represent it in a satisfactory manner by a normal interval. The set of dates which defines Recently is a fuzzy set.

A fuzzy predicate A is defined by a function f_A in a given domain (dates for example) in the interval [0, 1]. This function is called a *membership function*. In Figure 2.6, each point represents the membership degree (ordinate) of the predicate Recently for a particular date (abscissa), i.e.:

if $f_A(x) = 1$, then x satisfies A perfectly
if $f_A(x) = 0$, then x absolutely cannot satisfy A

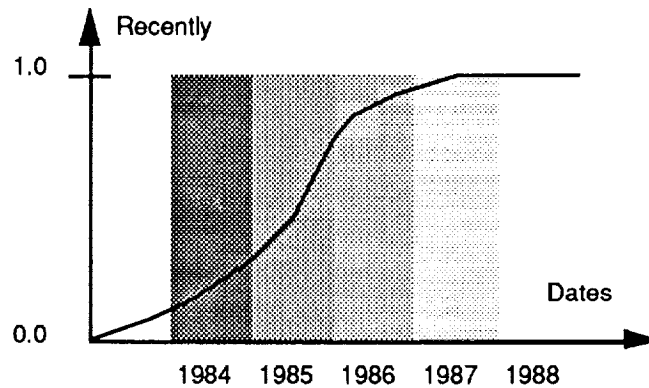


Figure 2.6. Membership function Recently.

Fuzzy sets and possibility

A fuzzy set (Dubois & Prade, 1980, 1985) can be used to represent a poorly defined constraint. Such a constraint implies a set of more or less acceptable values. Consider an event A (for example, A = Date of publication around 1977). A measure of possibility $\Pi(A)$ can be built based on a possibility distribution $\pi(x)$, where x is an elementary event, as follows:

$$\Pi(A) = \sup_{x \in A} \pi(x)$$

A possibility distribution¹⁰ represents a fuzzy set of more or less possible values for a variable. Consequently, one can interpret the possibility that the proposition X has the value x knowing that X is A, as the membership degree of x to A, i.e. $\pi(x) = f_A(x)$. Thus, a fuzzy set permits the definition of a set of more or less possible values for a variable limited by a poorly defined constraint.

2.1.5.4.2. Truth of a Proposition

In the context of intelligent assistance, the truth of a proposition is its conformity with the facts or reality as perceived by users.

Representation of an imprecise proposition

Consider the proposition: R = d is A (for example, this document was published recently), where d is a given subset of a set D (for example, Documents), and A is a predicate, sometimes vague (e.g. Published recently), which limits the possible values of d. The content of a proposition may be represented by a possibility distribution $\pi_{a(d)}$ associated with the function a(d) (for example, the date of publication of the document d). $\pi_{a(d)}$ is an application of a given set $X=\{x_i\}$ (for example, the dates of publication) in the interval [0, 1]. This is the membership function f_A associated with the predicate A:

$$\forall x \in X, \pi_{a(d)}(x) = f_A(x)$$

Fuzzy pattern matching

Consider two propositions: a query Q and a characteristic of a database a(d). The two propositions are represented by the possibility distributions π_Q and $\pi_{a(d)}$ respectively. Consider the interval $[N(a(d) | Q), \Pi(a(d) | Q)]$, where $\Pi(a(d) | Q)$ and $N(a(d) | Q)$ are the possibility and the necessity respectively that a(d) is true for a given query Q:

$$\begin{aligned} \Pi(a(d) | Q) &= \sup_{x \in X} \{ \min \{ \pi_Q(x), \pi_{a(d)}(x) \} \} \\ N(a(d) | Q) &= \inf_{x \in X} \{ \max \{ 1 - \pi_Q(x), \pi_{a(d)}(x) \} \} \end{aligned}$$

$\Pi(a(d) | Q)$ evaluates the degree of intersection of $\pi_R(x)$ and $\pi_{a(d)}(x)$. $N(a(d) | Q)$ evaluates the degree of inclusion of $\pi_{a(d)}(x)$ in $\pi_Q(x)$.

If the datum is precise, then $\pi_{a(d)}(x)$ is characterized by a possibility distribution of zero for all points except one. In this case, $\Pi(a(d) | Q) = N(a(d) | Q)$. This degree can be interpreted as a degree of truth.

2.1.5.4.3. Knowledge Representation

A piece of basic information can be represented by the triplet (attribute, object, value), where value is an element of the domain of the attribute under consideration, e.g.

¹⁰ The possibility of an event is calculated from the best case (in the possibility distribution), and not from the accumulation of cases which are more or less good, as in probability theory. As a result, the possibility of an event tries to evaluate its feasibility. If the possibility of an event A equals 1, it can occur that the possibility of $\neg A$ also equals 1. If the necessity of an event A equals 1 (A is certain), then the possibility and necessity of $\neg A$ equal 0. These results lead to the following theorem: $N(A) = 1 - \Pi(\neg A)$.

(date_of_publication, book_Y, 1972), (author, book_X, Dupont). The value of an attribute can be precise or imprecise. The possible values of an attribute can be represented by a possibility distribution on its domain. A precise value is represented by a possibility distribution of zero for every point except the point corresponding to that precise value, where it has a value of 1. An unknown value is represented by a possibility distribution equal to 1 for all points in the attribute domain.

The value of an attribute is often uncertain¹¹. This uncertainty is usually the result of a lack of confidence (of users) in the source of the information. Subjective evaluation scales are used to take this type of uncertainty into account. For example, the scale shown in Table 2.1 presents the association of various judgements on the subjective rating u . These ratings correspond to subjective probabilities and permit "modulation" of the initial membership functions which characterize only the imprecision. It should be noted that this type of scale may be expressed in the form of confidence intervals instead of simple ratings.

Table 2.1. Uncertainty rating scale.

Degree	Judgement	u
5	I am sure that the information is good	1.0
4	I think that the information is good	0.7
3	I don't know	0.5
2	I think that the information is not good	0.3
1	I am sure that the information is not good	0.0

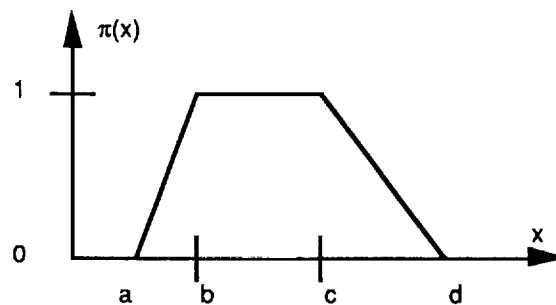


Figure 2.7. Possibility distribution for a quantitative variable.

¹¹ In speaking about uncertainty, two concepts are often mixed: imprecision on the one hand, and uncertainty proper on the other. The concept of imprecision relates to "that which is not precisely known": for example, the size of a physical device may not be known precisely, e.g., the pressure is between 40 mm Hg and 360 mm Hg, and the pressure is normal. A precise version of these expressions would be, for instance: the pressure is exactly 150 mm Hg; the pressure in tank 2 follows the equation of state of an ideal gas, i.e. $PV=RT$, where V is the volume of the tank, T is the internal temperature and R the universal gas constant. These statements are not necessarily true, however they are precise.

The concept of uncertainty proper is related to "that which is not necessarily true", e.g., it will rain tomorrow, and valve V4 leaks. We will say that information is certain when it is considered to be true. A fact, formalized as a proposition, is uncertain as long as its truth value is not proven.

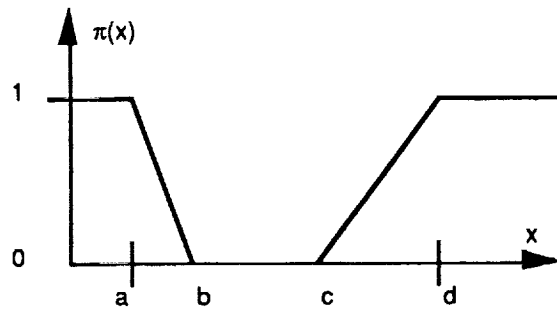


Figure 2.8. Possibility distribution: inverse representation.

Quantitative attributes

It is common to use a trapezoidal representation for the possibility distributions of quantitative variables. This type of representation is at the same time very simple, easy to use and robust. It is a good approximation to the common-sense notion of precision. It can be associated with the quintuplet (z, a, b, c, d) , where z, a, b, c and d are such that (Figures 2.7 and 2.8):

$[b, c]$	is the <i>preferred domain</i> ,
$[a, b] \cup [c, d]$	is the <i>permitted domain</i> ,
the complement of $[a, d]$	is the <i>unacceptable domain</i> , and
$z = 1$	for the <i>regular configuration</i> ,
$z = 0$	for a possibility distribution equal to 1 at all points,
$z = -1$	for an <i>inverse configuration</i> .

Uncertainty affects imprecision, i.e. the corresponding possibility distribution. We consider the following three observations.

1. When the uncertainty increases, the corresponding possibility distribution "expands".
2. When a proposition is absolutely uncertain (total ignorance), its possibility distribution is equal to 1 for all points in the domain.
3. When the certainty of the contrary proposition increases, it is necessary to consider the complement to 1 of the possibility distribution of the proposition under consideration.

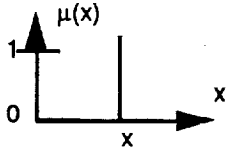
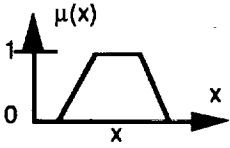
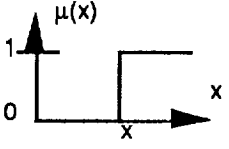
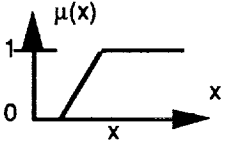
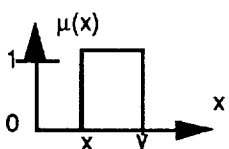
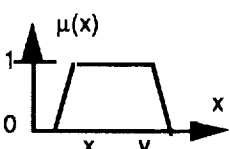
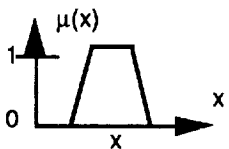
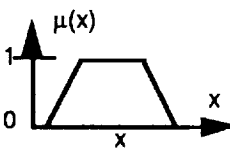
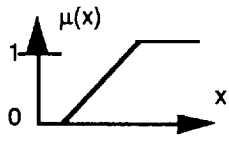
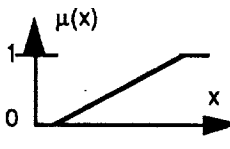
Let u be the subjective rating attached to a given proposition p and (z, a, b, c, d) the possibility distribution representing the imprecision of p . Taking into account the above observations, the following consequences may be inferred. The resulting possibility distribution (z', a', b', c', d') may be such that the preferred domain $[b, c]$ and the acceptable domain $[a, d]$ are enlarged by a certain coefficient, for example $1/(2u-1)$. In addition, we will have the following rules:

if $0 \leq u < 0.5$	then $z' = -z$,
if $u = 0.5$	then $z' = 0$,
if $0.5 < u \leq 1$	then $z' = z$.

For example, in the library management system BIBLIO, which we developed at the Toulouse Research Center (CERT, France), six options have been chosen for entering a

date (year): in X, after X, before X, between X and Y, around X and recently. Five of the corresponding possibility distributions are represented in Table 2.2 (after X and before X are symmetrical). Only the degrees of certainty 5 and 4 are represented, i.e. the degrees 2 and 1 are their respective complements to 1. Degree 3 corresponds to a possibility distribution equal to 1 in all cases.

Table 2.2. Examples of the possibility distributions represented in BIBLIO.

Degree of certainty 5	Degree of certainty 4	
		in X
		after X
		between X and Y
		around X
		recently

Qualitative attributes

The possibility distribution associated with a qualitative attribute is represented by a set of attribute pairs $\{ (\{x_1\}, t_1), \dots, (\{x_n\}, t_n) \}$ (see Figure 2.9).

The subsets $\{x_1\}, \dots, \{x_n\}$ are characterized respectively by the discrete possibility distributions (t_1, \dots, t_n) . This representation is conceptually simple and easy to manipulate. In the library management example, the qualitative attributes could be, for example, the names of authors or keywords. A request from a user might be expressed in the following manner:

x is A_1 or A_2 ... or A_n .

For example, "I think that the name of the author is Dupond (1.0), but it could also be Durand (0.6) or Smith (0.4), or someone else (0.1)".

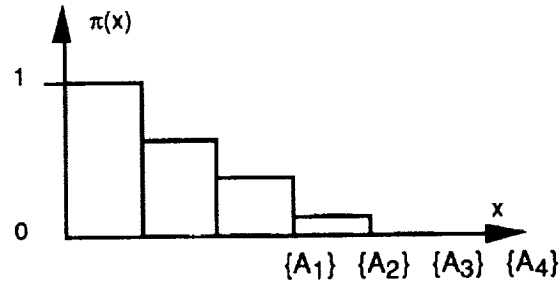


Figure 2.9 . Possibility distribution for a qualitative variable.
In the example, $\{A_4\}$ is identical to $X - \{A_1, A_2, A_3\}$.

If X is the domain of the attribute, i.e. the list of authors, then this request takes the following form:

$$\{ (\{A_1\}, 1.0), (\{A_2\}, 0.6), (\{A_3\}, 0.4), (X - \{A_1, A_2, A_3\}, 0.1) \}$$

The attributes may have either a well-defined value or an unknown value. If x is an attribute and A a value of x , then two types of propositions are possible in the database:

x is A , for example the name of the author is Smith

which is represented by the possibility distribution:

$$\{ (\{A\}, 1), (X - \{A\}, 0) \}$$

and

x is unknown

which is represented by the possibility distribution:

$$\{ (X, 1) \}$$

It should be noted that the two propositions x is A and x includes A are represented by the same possibility distribution.

The uncertainty of the user can be taken into account as follows. If u is a subjective probability associated with a given proposition " X is A " and $\{(\{A\}, 1), (X - \{A\}, 0)\}$ is the possibility distribution representing the precision of " X is A ", then the resulting possibility distribution might be the following:

$$\begin{array}{ll} \text{if } 0 \leq u < 0.5 & \{ (\{A\}, u), (X - \{A\}, 1) \} \\ \text{if } u = 0.5 & \{ (X, 1) \} \\ \text{if } 0.5 \leq u < 1 & \{ (\{A\}, 1), (X - \{A\}, 1-u) \} \end{array}$$

Examples:

1. Let the proposition the publisher is CERT have a degree of certainty corresponding to a subjective probability of 0.4. The possibility distribution associated with this proposition is:

$$\{ (\{CERT\}, 1), (X - \{CERT\}, 0.4) \}$$

where X is the domain of the attribute publisher.

2. Let the proposition the author is Hugo have a degree of certainty corresponding to a subjective rating of 0.0 (the user is sure that the name of the author is not Hugo). The possibility distribution associated with this proposition is:

$$\{ (\{Hugo\} , 0), (X - \{Hugo\}, 1) \}$$

where X is the domain of the attribute name of author.

2.1.6. Conclusions

What we called conventional approaches to indexing and information retrieval base their performance according to measures of recall and precision. They do not include costs involved in incremental search at a terminal. In particular, user-friendliness of the interface is likely to improve the cognitive orientation of the search. They are all rigid and very dependent on their own normative model. Furthermore, none of these methods are context-sensitive. However, they can constitute very good startup procedures for more knowledge-based approaches to indexing and information retrieval.

2.2. HyperText

2.2.1. History and Definitions

Vannevar Bush, who was President Roosevelt's science advisor, envisioned in 1945 new information organization and retrieval concepts which led to a machine called a *Memex*. The idea was to allow anyone to browse and make associative links between any references in a library. Douglas Engelbart was the first scientist influenced by Bush's concepts. Engelbart's research at the Stanford Research Institute (SRI) was centred around the augmentation of the human intellect (Engelbart, 1963). In the early 1960s, he began to develop the on-line system NLS. This system was renamed Augment when Engelbart was at McDonnell-Douglas. Augment is an on-line work environment. In its original form, it served as: a storage system for memos, research notes and documentation; as a communications network, since on-line conferencing was possible; and as a shared workspace where researchers could plan and design projects. In order to browse faster in Augment, Engelbart invented the mouse as an input device. Ted Nelson coined the word *HyperText* to mean nonsequential writing with free user movement along links (Nelson, 1967). Advanced electronic publishing was born with the Xanadu system developed by Nelson (1988).

At its most basic level, HyperText is a database management system (DBMS) that lets users connect screens of information using associative links (Fiderio, 1988). It is a combination of natural language text with the computer's capacity for interactive branching, or dynamic display of nonlinear text which cannot be printed on a conventional page (Nelson, 1967).

2.2.1.1. Linear text

When reading a book page after page, the contents are scanned in a linear manner. Similarly, if a programmer wishes to insert a useful comment in a program, it is usual to

insert the comment after a line of code. In this way, LISP code and comments can be mixed, i.e. two types of information are mixed at the same level:

```
(defun hypertext ()           ;; will provide a definition for
                             ;; HyperText when clicked
  (cond (mouseClick selfZone) (print comment)))
```

When accessing a file on a computer, a hierarchy of directories is used which generates a linear path. For instance, if users need to go from Text-1 to Text-2, they have to backtrack to the Directory level and go down to Text-2 (Figure 2.10). In other words, there is no direct link possible between Text-1 and Text-2.

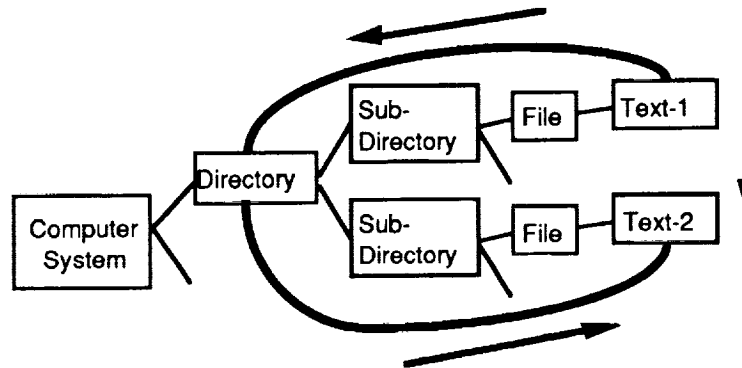


Figure 2.10. Linear access to a text using a classical computer operating system.

2.2.1.2. Nonlinear text

In the above example of LISP code mixed with comments, an alternative using a HyperText system would be to allow the programmer to click on any word of the LISP code and for example create a window for the comment. This approach also allows modularity. Furthermore, in the example presented in Figure 2.11, if users need to go from one text to another which is not in the same directory, in the linear approach they will have to backtrack and find a new linear path in the hierarchy allowing them to go to the desired text. In the nonlinear approach, users simply build a link between the current text and the target.

HyperText is also called *nonlinear text*. The concept of HyperText is quite simple: windows on the screen are associated with objects in a database, and links are provided between these objects, both graphically and in the database (Conklin, 1987). A HyperText system may be described as a system including a database which is a network of textual or graphical nodes, and windows on the screen corresponding one-to-one with nodes in the database. The HyperText database is a directed graph. Note that a small number of nodes are open on the screen at any one time.

2.2.1.3. What a HyperText system is not

Conklin distinguishes HyperText from other computer software by pointing out what it is not. It is not a *window system* like *Windows*, which has no underlying database. It is more than a conventional *file system* which does not have the sophisticated link concept

available in HyperText systems. It is not an *outline processor*, which provides no support for references between outline entries. It is not a *text formatting system*, which is purely hierarchical and does not provide any mechanism for navigating within a document. It is not a conventional *database management system (DBMS)*, which does not provide any user interface associating objects in the database to objects on the screen.

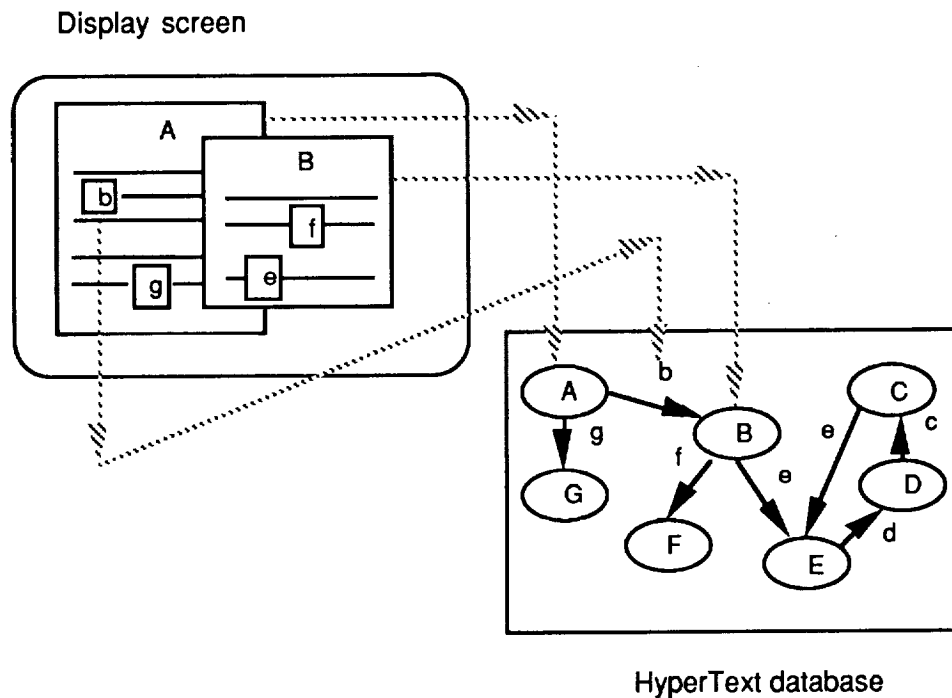


Figure 2.11 . Association of displayed objects with objects in a database (Conklin, 1987).

2.2.1.4. Browsing

Browsing is the navigational mechanism provided by HyperText. A HyperText document can be browsed either by clicking on any screen object, following the links and opening windows, by searching the network for a string (information retrieval) or by using an overview network (Figure 2.12) of the HyperText document and navigating with it (like navigating in a country by using a map). In Figure 2.12, for example, users are provided with an explicit map telling them that if they are watching node B (e.g. a page of text), then they can get more information by clicking on the descriptor f and get the node F (e.g. a graphics display), or on the descriptor e and get the node E (e.g. a text definition of the descriptor e).

2.2.1.5. Implementation

On-line reference manuals and documentation were the first types of applications for HyperText. The main goal of Bush's Memex system was to mechanize the scientific literature (Bush, 1945). Memex used microfilms and photocells. It was a very large library

of notes, photographs and sketches. Bush built a link mechanism to go from any point to any other point in Memex.

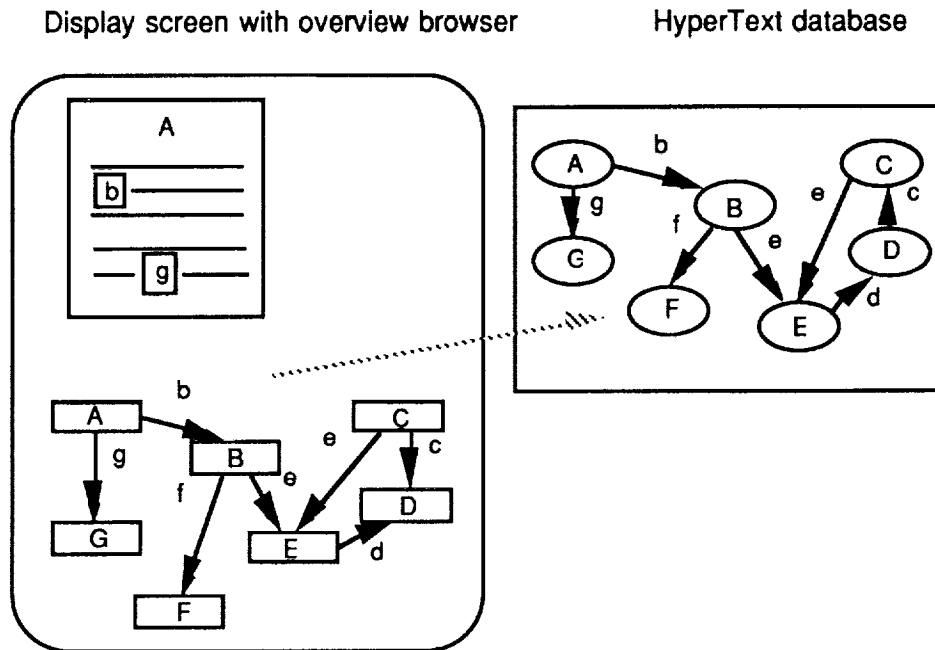


Figure 2.12. Browsing through a HyperText using an overview network.

Public information systems, such as the Medical Handbook developed by Mark Frisse and coworkers, show the important problem of information retrieval in large HyperText systems. Retrieval requires a combination of browsing and full-text document retrieval techniques. Frisse's work shows that indexing remains one of the most difficult problems (Friss, 1987; Frisse & Cousins, 1989). Vannevar Bush wrote that "our ineptitude in getting at the record is largely caused by the artificiality of the system of indexing". Trigg's thesis on the *Textnet* concerned a "network-based approach to text handling for the on-line scientific community" (Trigg, 1983). Note that electronic mail has been a major factor in the recent public success of HyperText.

Authoring systems constitute another class of HyperText applications. These systems are designed to help users formalize complex concepts. For instance, there are authoring systems for editing tutorials or specifications of systems to be designed. Horst Rittel and his students (Rittel & Webber, 1973) introduced the concept of issue-based information systems (IBIS) to solve problems that cannot be solved by traditional systems analysis. IBIS systems are a combination of teleconferencing systems and HyperText. IBIS systems have three types of nodes: issues, positions and arguments. They use nine types of relations to link these nodes: *Is_suggested_by*, *Responds_to*, *Questions*, *Supports*, *Objects_to*, *Specializes*, *Generalizes*, *Refers_to* and *Replaces* (Figure 2.13).

Cooperative work systems were introduced by Douglas Engelbart, who was interested in augmenting the human intellect. His NLS system was designed as an experimental tool which allowed people to store specifications, plans, design, programs, documents, reports, memos, bibliography and reference notes, etc., and do outlining, planning, designing, debugging and "a good deal of our intercommunication, via the consoles" (Engelbart, 1963). This was a forerunner of office automation and electronic mail systems. Designing systems for augmenting the human intellect is a closed-loop process between design and

operations. Rapid prototyping is very important to this process. In a similar vein, Engelbart is currently promoting the idea of *computer supported cooperative work* (CSCW) with his bootstrapping project at Stanford University.

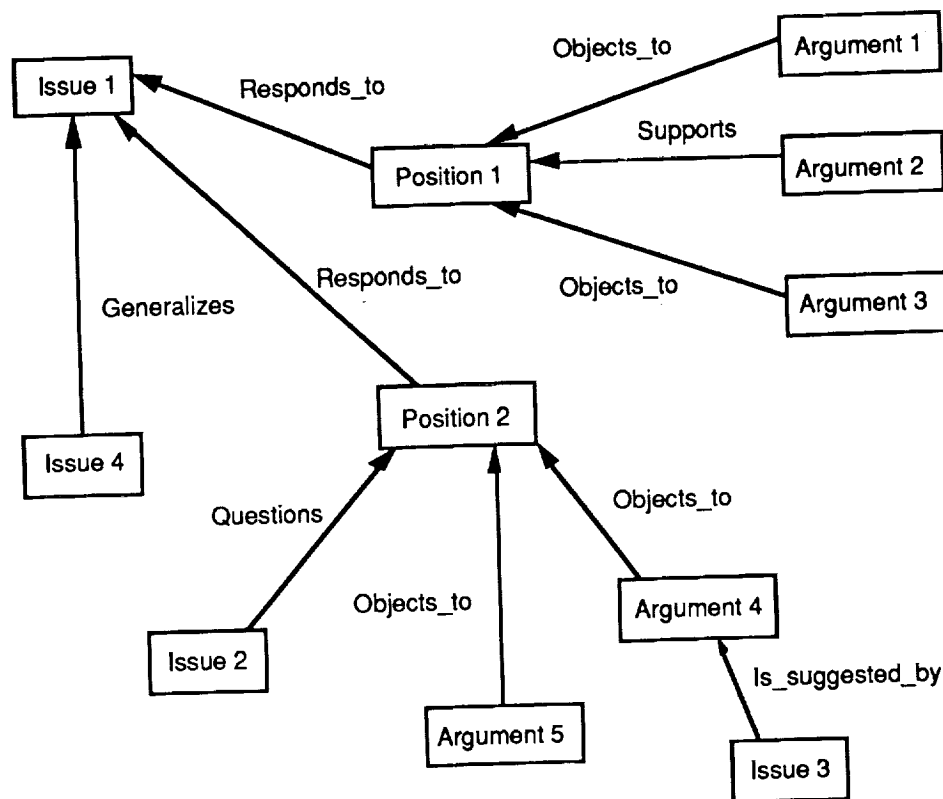


Figure 2.13. Example of a topology of an IBIS network.

Browsing systems such as ZOG, developed at Carnegie Mellon University, involve large databases of small segments which can be displayed on a screen (McCracken & Akscyn, 1984). The commercial version of ZOG was called knowledge management system (KMS). Each segment of KMS is a frame, as shown in Figure 2.14 (Akscyn *et al.*, 1987). In 1982, KMS was installed and used as a computer-based information management system on a nuclear-powered aircraft carrier.

The University of Maryland has developed the Hyperties project to get an experimental environment to study HyperText and a product which is a documentary about Austria and the Holocaust (Shneiderman, 1987b). Hyperties runs on an IBM PC with or without touch screens. Document Examiner provided by the Symbolics company is certainly the most advanced of the on-line help systems (Walker, 1987). Unlike many HyperText interfaces, Document Examiner does not adopt the directed graph (HyperText database) as its fundamental user-visible navigation model. Instead it offers context evaluation and context-based searching capabilities that are based on consideration of the strategies that people use in interacting with paper documents.

The best known HyperText is probably the NoteCards system developed at Xerox PARC (Halasz *et al.*, 1987). NoteCards is an information analyst's support tool. NoteCards allows the gathering of information and production of analytical reports. It is written in LISP and is an open architecture that allows new applications to be built on top of it. Its primary purpose is experimentation with HyperText.

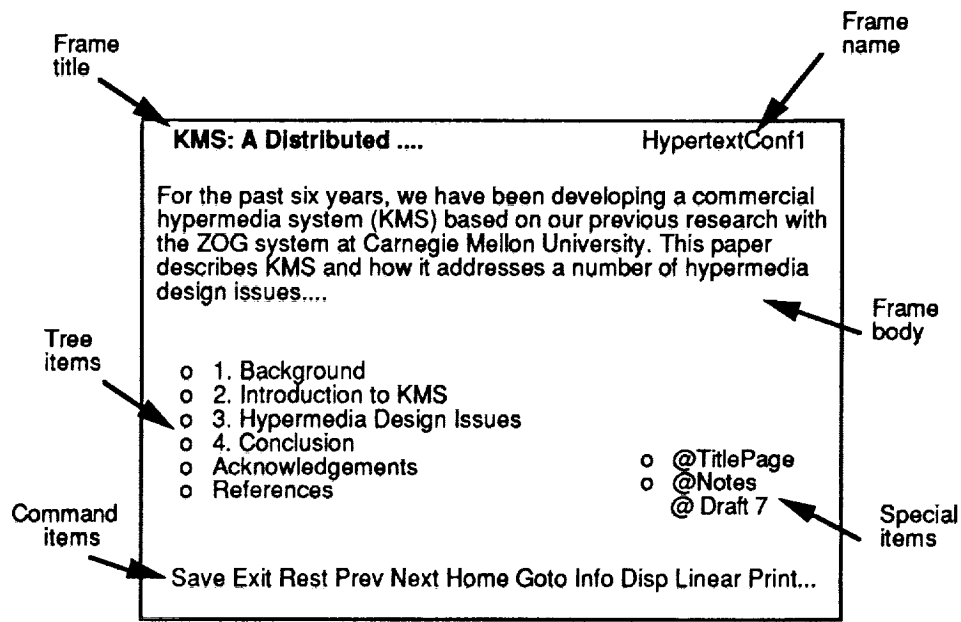


Figure 2.14. Example of a KMS frame.

The Institute for Research in Information and Scholarship (IRIS) at Brown University is one of the oldest HyperText centres. The Intermedia project was developed both as a tool for professors to organize and present their lesson material via computer, and as an interactive medium for students to study the material and add their own annotations and reports (Yankelovich *et al.*, 1985). Intermedia provides tools such as text, graphics and timeline editors, a 3D viewer and a scanned-image viewer. A new construct, called a web, was introduced in Intermedia for the implementation of context-dependent link display. The Intermedia browser has two kinds of displays: a global map (for global navigation) and a local map (for local navigation within a single document).

Tektronix HyperText Abstract Machine (HAM) is a general purpose, transaction-based server for a HyperText storage system (Campbell & Goodman, 1987). The server is designed to handle multiple users in a network environment. The storage system consists of a collection of contexts, nodes, links and attributes that make up a HyperText graph. Several of the HyperText architectures described above can be implemented easily using the HAM's storage model.

The most commonly used HyperText system is undoubtedly HyperCard distributed by Apple on the Macintosh. We will take HyperCard as an example and describe it in Appendix D.

Applications of HyperText systems include electronic publishing, on-line technical manuals, on-line instruction manuals, on-line help for other software, project management, issue analysis, on-line policy manuals, group presentations via large-screen projectors, financial modelling, user interfaces to videodisc-based materials, user interfaces to expert systems, software engineering, computer-assisted foreign language translation, operating system shells, engineering design and research in general (Akscyn *et al.*, 1987). For example, there is currently a research project at Xerox PARC on HyperText and knowledge representation to design an instructional design environment (IDE), i.e. an environment for designing teaching tools. HyperText features in the IDE software are used to capture and organize into one coherent framework the entire corpora of theoretical, instructional and content material that affect foreign language instruction (Jordan *et al.*, 1989).

2.2.2. From Text to Hypertext

An alternative to manual composition of HyperText databases is conversion from existing text, millions of pages of which already exist in industry and government agencies. Moreover, electronic mail is constantly increasing and produces more and more text information to process and reorganize. Such conversion often requires analysis of the text document in order to determine how best to represent its structure. Generally the intentions of the authors are not at all clear in the flat text (as opposed to the HyperText). The medium used for delivering a document can greatly influence the way in which the document will be perceived and understood. Furthermore, authors generally use the medium to improve the clarity of what they want to convey. Raymond & Tompa (1987) analysed this problem of conversion on the *Oxford English Dictionary*. They observed that a document can be broken into many networks of arbitrarily chosen pages or nodes, but not all of these are suitable representations of the whole content of the original text. HyperText nodes are *text fragments* which have the special characteristic of being both semantically and syntactically discrete. Ideally, each node represents a single, independent concept which is a candidate for classification. However, not all concepts are representable in this fashion. In this case, syntactic fragmentation may lose semantic information. The fact that links are also discrete and explicit tends to lead to a loss of text richness, where links between concepts are implicit. In other words, the way a document decomposes into nodes and links will affect the overall content of the document.

Conversion from text to HyperText makes the *implicit* structure *explicit*. To illustrate this conversion process, we will take an example of the conversion of the NASA Documentation for the Space Station "Freedom" from the original paper documents to a HyperText system. Although this work is still a research project, it illustrates some of the possibilities offered by HyperText technology. The HyperText product is called *Computer Integrated Documentation* (CID) (Boy, 1989b). The idea of integration is essential. It covers the integration of information available in different modes (text, graphics, sound, images) and includes the concept of context of search (integration of the user). It also covers the incremental construction of system documentation from design to the final operation of the system (integration of knowledge). The process described can be adapted easily to any other technical documentation.

2.2.2.1. Description of an example of paper documentation

The complexity of modern engineered systems stresses the need for good documentation systems. Such documentation is very labour-intensive to manage and maintain, requiring a great deal of expertise and development time. It is important to note that this documentation is most useful and necessary during the actual operations of the systems that it describes. Furthermore, most documentation systems (generally paper-based) are not self-explanatory because users need cross-references and so his desktop quickly becomes covered with documents.

Designing a complex system like the Space Station is an iterative process. Its documentation system is designed to handle a huge amount of information and is organized as follows. There is a Program Requirement Document (PRD) which is the central document of the document network. It establishes the highest-level requirements associated with the Space Station Program. Generally, the other program documents develop the PRD topics in more detail. Each document, including the PRD, has a baseline issue and revisions. Indeed, technical documentation is developed incrementally. A new directive may specify changes to a particular document, which leads to a revision. Based on this analysis, two dependency graphs can be drawn: the development dependency graph and the revision dependency graph. The *development dependency graph* is a goal-oriented structure

(Figure 2.15). A document at level n specifies goals to be developed in documents at level $n+1$. In the present example, the PRD defines level zero of the development dependency graph. The *revision dependency graph* is a history-driven graph (Figure 2.16) and describes the history of revisions of a given document.

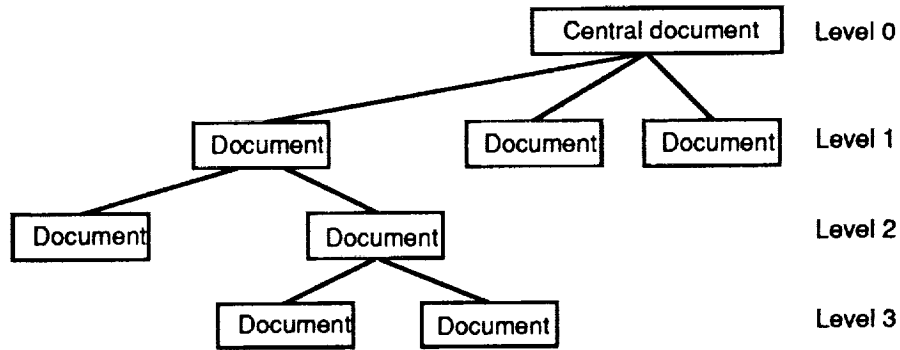


Figure 2.15. Development dependency graph.

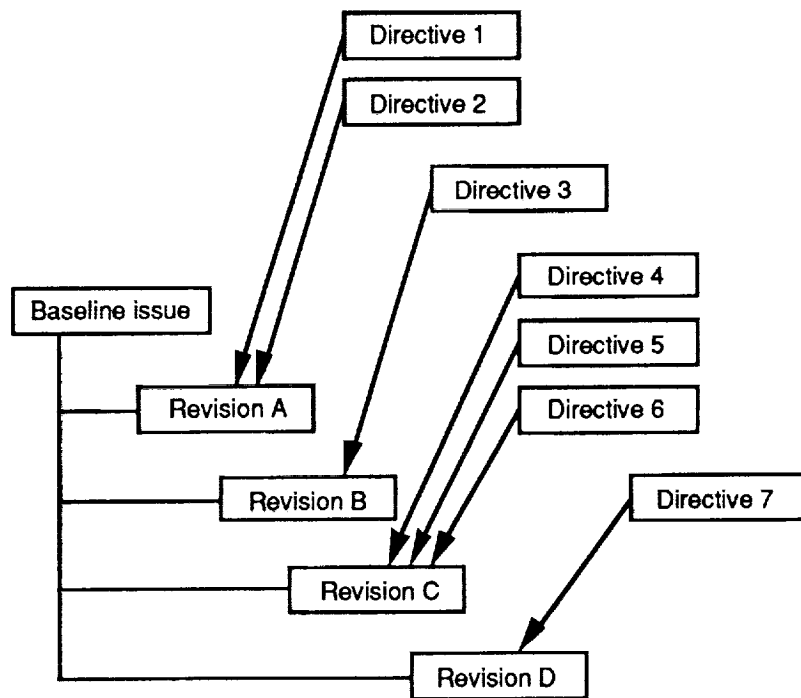


Figure 2.16. Revision dependency graph.

Functional fragmentations of paper documents are made according to the task involved. Each document includes the following major nodes: a description of the document (baseline issue, revisions, directives), a preface, a table of contents, a body of text segmented into sections and subsections, an abbreviations table, a definitions table and appendices (Figure 2.17). We will call a "macro" link a set of links between two major nodes. Each major

node may contain a hierarchy of nodes. For instance, in the body of text, there is a hierarchy of sections. There are links between sections which are linear (section to next section) and nonlinear (reference to a section other than the next one). There are references to other major nodes within a document (the context of the document is retained) and sometimes to other documents (the context of the document is changed to that of another document).

2.2.2.2. Modelling the nodes and the links

Nodes include part of the original text (including graphics). Links may have various meanings: *abbreviation-of*, *definition-of*, *development-of*, *go-to-section*, *show-graphics-of*, *find-information-on*, etc. The links are also called *actions* because they are activated by users when they are browsing the CID. Links are built as follows. A script is attached to any part of the text (or graphics) which is recognized as a relevant *descriptor*, the concept of which is very important in information retrieval (Salton, 1989). A descriptor is defined as any word or phrase (or piece of graphics) which will provide a rich *precondition* for a search in the documentation HyperText database.

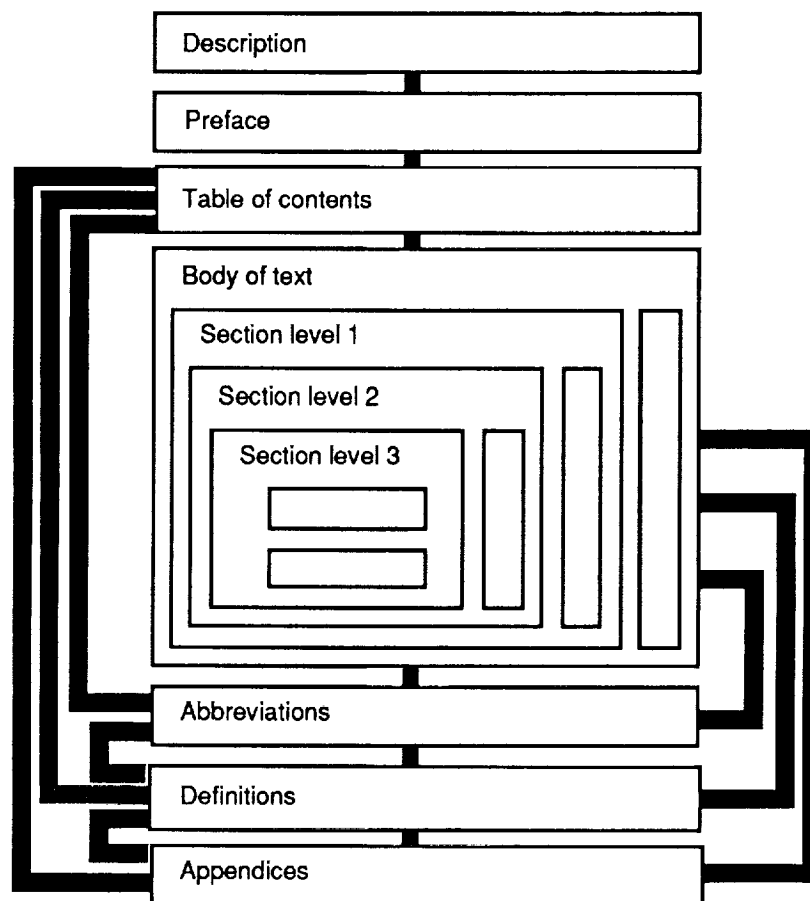


Figure 2.17. Description of a document (main nodes and "macro" links).

Building such descriptors requires expertise in the domain of investigation. We call a *referent* any part of the text which is linked to a descriptor. In a regular book, the table of contents provides a list of descriptors (section titles). These descriptors are organized hierarchically and have a one-to-one mapping with the corresponding pages. Unfortunately, in general, a descriptor may be linked to several referents. Thus, users may not find the relevant information in the first referent found. An example of this can be found in any index at the end of a book. The index is a list of descriptors followed by a list of numbers (pages) indexing the referents in the text. For instance, "word-1, 56, 177, 306" means that the "word-1" can be found on page 56, page 177 and page 306. The role of indexing is to bind descriptors to appropriate locations in the document. In the CID, any part of the document can provide descriptors which are linked to other parts of the document or to other documents. Generally, beginners (people who are not familiar with the book being explored) browse through the book using the table of contents (hierarchical browsing). As they become more knowledgeable about the contents of the book, they are able to understand the meaning of some of the descriptors available in the index. Thus, they can begin to use the index. If they are using the index for the first time, they do not have any reason not to select the first page provided. They go to this page. If the corresponding information is relevant, they are satisfied and the retrieval process is finished. If the corresponding information is not relevant they generally go back to the index. It would then be rational to try the second page proposed. This is a *backtracking* process. People are not good at backtracking. They tend to build context around the descriptors already used for the next time! For instance, they tend to remember that a descriptor was successful in a particular situation, although this memory does not generally last weeks or months (conversely, it does in a machine.)

2.2.3. Conclusions

Hypertext technology is very interesting to develop appropriate interfaces to documentation systems. It provides non-linear access to information. However, hypertext systems are not providing any indexing mechanism that allows users to retrieve the right information at the right time. For this reason, another technology must be implemented to bridge this gap. As search in documentation is very dependent on the knowledge users have on the way to retrieve such or such information in such or such context, knowledge-based techniques are presented as possible solutions in the next section.

2.3. Knowledge-Based Systems

Knowledge approaches to indexing are generally known as *concept-based search*. One challenge is to understand how to relate useful concepts to pieces of information. Indexing becomes the process of conceptualization of documents, i.e., building conceptual descriptions (that can be called descriptors) of appropriate documentation fragments (referents). The main difference with conventional techniques will be that descriptors will be semantically linked between each other. Thus, inference from concept to concept will be possible.

Information retrieval in a document can be seen as problem solving. In artificial intelligence (AI), problem solving is generally represented as search in a problem space. Many systems have been developed that search problem spaces by applying operators and expanding new states until they find a path between an initial state (query) and a final state (desired referent). In information retrieval, the main problem is that the final state (or goal) is not always well formalized. Users are the unique judge to declare if the goal has been reached or not. Furthermore, a good information retrieval system must be able to retrieve

information without letting the amount of retrieved information become too large. The underlying mechanism involves constraint-based search. The AI and Cognitive Science community has been recently focusing on the development of computational models of human problem-solving behavior (Newell & Rosenbloom, 1981; Anderson, 1983; Ohlsson, 1987).

Other attempts have been made in the area of natural language processing. In particular, Quillian (1968) introduced the spreading-activation model that allows the representation of connections between concepts. Once a concept is activated, it is marked, and all the semantically related concepts become activated also ("marker passing" in a semantic network). In his ACT framework, Anderson (1976, 1983) augmented the characteristics of the semantic network by introducing the notion of trace strength on the links, and a numerical level of activation on each node. This type of memory model has been taken recently by Jones (1989) as a support for his model of retrieval in problem solving.

Other AI approaches to documentation information retrieval are based on cognitive models of librarians' retrieval strategies. As Chen & Dhar (1987) mention, to make large information banks more accessible by computer, it is best to first try to understand how reference librarians actually help users, and then to try and include these capabilities in on-line systems. The major problem in incorporating a model of user-librarian interactions into the system is the difficulty of acquiring the information for this model.

2.3.1. Concept Indexing

2.3.1.1. Concept hierarchies acquisition from documentation experts

It is possible to acquire concept hierarchies directly from documentation experts. They generally reproduce a content analysis to extract relevant concept descriptors and their relations. At this point, the methods¹² described in section 2.1 can be used to initiate the knowledge acquisition process as well as to normalize expert knowledge. Descriptors can be recursively grouped into classes. Thus, if each referent r_i described by a set of concept descriptor d_j , d_j can be given by users to retrieve r_i or inferred from another concept d_k semantically related to d_j . Concept hierarchies can be acquired manually or interactively with an assistant system.

Interactive indexing techniques have been developed already. For instance, at the National Library of Medicine, a frame representation of document concepts has been used in the Indexing Aid System (Humphrey, 1987). The corresponding system includes inheritance, restrictions, and various functions implemented as procedural attachments on the concept frames.

2.3.1.2. Concept indexing by end-users

There are at least two good reasons to get indexing concepts from end-users. The first reason is that concept hierarchies given by documentation experts can be useful but are not necessarily personalized to users. In other words, users should be able to easily annotate their documentation, and consequently modify provided indexing means. The second reason is that it is only when people use the documentation that they realize what information is useful (or not useful) in a given situation. In other words, indexing concepts are difficult to anticipate, but they can be very easy to formulate when users face the corresponding information. We must notice that concept indexing by end-users is a

¹² Conventional approaches to indexing and information retrieval.

complementary process to the concept hierarchies acquisition from documentation experts. Furthermore, both processes can be circularly performed. In chapter 3, we will develop a theory based on the assumption that descriptor-referent, descriptor-descriptor and referent-referent semantic relationships are context-dependant.

2.3.2. Text Analysis and Interpretation

If we assume that the text to be processed is content-rich, i.e., there are explicitly in the text words or phrases that can be taken as partial knowledge to build a knowledge base representing the text. Parsaye et al. (1989) have proposed four types of data structures to store and use resulting knowledge: lists, frames, relational databases, and hierarchies. Lists constitute the most basic structures. Frames introduce the concept of slots in list elements. When the same set of slots is used for all referents a relational table can be built. This allows a variety of operators to be applied in accessing the information within referents.

The main task in the development of a text interpretation system is the translation of documentation information into usable knowledge. During the first phase, one extracts all the significant descriptors (included in the text) using a "stop list" that includes all the words and expressions that do not characterize the domain. Obviously, this task is incremental by nature, i.e., the stop list is updated incrementally. To discover new expressions, a list of domain-dependant descriptors can be maintained also. In a second phase, an expert can be used to structure the list of domain-dependant descriptors. A hierarchy of descriptors and synonyms should come up from this process.

The particular combination of descriptors in a query establishes a context (cf. Gestalt Psychology) that is more than the sum of the separate effects of each descriptors taken alone (Parsaye et al., 1989). Construction of these context is not a trivial task. A method to build a context is to capture local contexts and generalize them. For instance, take the four following queries:

1. (ENGINEERS) (SPACE SCIENCES) (RESEARCH)
2. (ENGINEERS) (INFORMATION SCIENCES) (RESEARCH)
3. (ENGINEERS) (SPACE SCIENCES) (DEVELOPMENT)
4. (SECRETARIES) (SPACE SCIENCES)

Parsaye et al. propose a process of context formation for each query that consists of three steps:

1. Find an appropriate descriptor class in the structured list of domain-dependant descriptors, i.e.,

(ENGINEERS)	belongs to	(professional category)
(SPACE SCIENCES)	belongs to	(organization)
(RESEARCH)	belongs to	(type of work)

2. For each descriptor instance in the query establish a context that includes the classes of the other descriptor instances in the query, i.e.,

((ENGINEERS)	(CONTEXT ((organization) (type of work))))
((SPACE SCIENCES)	(CONTEXT ((professional category) (type of work))))
((RESEARCH)	(CONTEXT ((professional category) (organization))))

3. For each "contextualized" descriptor, rebuild corresponding instances of contexts, i.e.,

((ENGINEERS)	((organization) ((SPACE SCIENCES))) ((type of work) ((RESEARCH))))
((SPACE SCIENCES)	((professional category) ((ENGINEERS))) ((type of work) ((RESEARCH))))
((RESEARCH)	((professional category) ((ENGINEERS))) ((organization) ((SPACE SCIENCES)))

Associations between descriptors and referents can be represented by rules such as the following:

If the user is an engineer doing research in space sciences
and asks for information on computer languages
then propose referents related to FORTRAN.

2.3.3. Conclusions

Current knowledge-based contributions to documentation indexing and information retrieval are focusing on concept indexing, and text analysis and interpretation. Both approaches are very young and not yet stabilized into formal theories. However, it seems clear that they bring more flexibility in the implementation of appropriate models of domains and tasks to be tackled. The former leads to concept clustering, a method that has been already developed in machine learning, in particular in the COBWEB system (Fisher, 1987). The latter leads to the application of natural language understanding paradigms.

2.4. Synthesis

The main technological objective of this project is to combine these three approaches (conventional indexing, hypertext, and knowledge-based systems) to develop appropriate semantic context-sensitive indexing and information retrieval.

Because of their associative capabilities, hypertext systems can be considered as mimics of the brain's ability to store and retrieve information by referential links for quick and intuitive access. Multipurpose research environments have been developed that encourage cooperative thinking on shared projects. Recent results have shown that hypertext is attractive for the development of large documentation systems (Glushko, 1989). However, although hypertext systems increase accessibility, they do not provide any built-in selectivity mechanism, i.e., you may have several candidate referents for one descriptor. Hypertext systems are also known to have problems:

1. *disorientation*, i.e., the tendency to lose one's sense of location and direction in a nonlinear document, e.g. if you are exploring a country where you have access to any location but without map; and
2. *cognitive overhead*, i.e., the additional effort and concentration necessary to maintain several tasks or trails at one time (Conklin, 1987).

In other words, while non-linear or hypertext systems may dramatically increase the accessibility of information, this increased accessibility may magnify an already severe problem of selection (Jones, 1987). A solution is to build appropriate indexing which will facilitate the process of information selection.

Conventional indexing methods have been described in section 2.1. They are systematic and can be automated. However, they are not very flexible and are very dependent on their

own underlying normative model. They can be improved by introducing knowledge into them. We will describe in the next chapters an AI approach to indexing. It must be noticed that some normative models of indexing can be kept and mixed into the knowledge-based approach we have chosen.

Our basic idea is based on the observation that information retrieval leads to the application of *search procedures*. For these reasons, knowledge-based systems technology can be very helpful for alleviating the selection problem and cognitive overhead of users. Another reason for developing a knowledge representation on top of (or imbedded in) hypertext systems is an easy mapping between the two: i.e., our knowledge representation can be naturally associated with hypertext nodes. This approach leads to the acquisition of context-sensitive descriptor-referent links. It is different from techniques like TCS (Text Categorization Shell) that analyze text at a conceptual level, without attempting a full analysis of meaning (Hayes et al., 1990). The technique that we are currently developing is based on the incremental modification (provided by users) of the descriptors semantically linked to referents.

Chapter 3

Contextual Knowledge

People use descriptors to retrieve information. These descriptors can be either explicit, e.g., table of contents or index, or implicit, e.g., a cognitive representation of the documentation that provides relations between pieces of information and approximative locations of them in the documentation. Users build cognitive maps to navigate in their documentation. Such cognitive maps are context-dependent. They are not the same from one user to another for the same documentation. They remember that this particular piece of information was (or was not) very interesting in such or such context.

Generally, indexing is done by book designers and not by users. An index and a table of content are generally presented. Unfortunately the descriptors printed are context-free. Because a descriptor can describe many referents, the problem of decidability introduces a major problem of backtracking that users may not accept especially when he is dealing with real-time operations. For such reasons, we have observed that technical and operational documentation is always fine-tuned by incrementally taking into account various situations (or context) in the indexing mechanism.

The problem is then to "contextify" the links between documentation nodes, i.e., relations between descriptors and referents. This will reduce the number of possible referents for a descriptor. This section presents a technique that solves this problem.

In this technical memorandum, the context acquisition problem is defined as discovery of abnormal conditions and generation of recovery actions, as well as reinforcement of current actions. Our implementation of a solution consists of two steps. It first observes users' decisions to add abnormal conditions or to reinforce current actions. If an abnormal condition has been isolated, the system either allows users to formulate a recovery action (using a text editor-like interface), or, if possible, records the subsequent users' strategy (i.e., the trace of his actions) for later analysis and formulation of a recovery action from this trace. For our documentation application (described in chapter 4), it is possible to record the trace of users' inputs (e.g., mouse clicks or keystrokes) in the electronic documentation. In the second step, if necessary, indices are refined into a more useable form according to context. This technique is developed in section 3.3.

3.1. Representing Contextual Knowledge

3.1.1. Introducing the Block Representation

The quality of the communication between two individuals most often depends on their reciprocal understanding of the internal model of the other. For example, a discussion between experts of the same domain will come down to an operative language (Falzon,

1986) which is highly "situational". In this case, the experts have quasi-identical knowledge of the subject they are talking about, in other words, their internal models are almost identical. In contrast, when a professor is addressing his students, each has a very different internal model. The professor must "decompile" his situational knowledge to make it intelligible to beginners. We will say that he uses an "analytical explanation" to make himself understood.

This distinction between analytical and situational is not new. In their critique of artificial intelligence, Hubert Dreyfus (1979) and Stuart Dreyfus (1982) underline the importance of situational knowledge in expertise, and the difficulty in eliciting and representing such knowledge so that it can be manipulated by a computer program. According to Hubert and Stuart Dreyfus, current expert systems can only represent the knowledge of a competent beginner, i.e. repeat the course given by the professor, or, in the best case, make a start at a few simple exercises.

The ideas in this section have evolved from several projects concerned with human-machine interaction and artificial intelligence. The first, the MESSAGE project, was a system designed to evaluate aircraft cockpit configurations by simultaneously monitoring pilot and aircraft behaviour and relating these to specific performance criteria. This necessitated building a human operator model (Boy & Tessier, 1985). We used it to analyze different types of errors and to demonstrate the utility of the decomposition of knowledge into analytical and situational forms. A second project undertaken in cooperation with NASA involved analyzing operator/system interactions in the task of fault diagnosis on the orbital refuelling system of the space shuttle (Boy, 1986a, 1987b). Finally, the block representation was initially developed in a third study focusing on operator assistance in telemanipulation (Boy & Delail, 1988; Boy & Mathé, 1988, 1989).

An index is represented as a *block* containing a description of a set of preconditions (triggering preconditions and contextual conditions), a set of postconditions (goals and abnormal conditions), and a set of actions to achieve the goals (Figure 3.1).

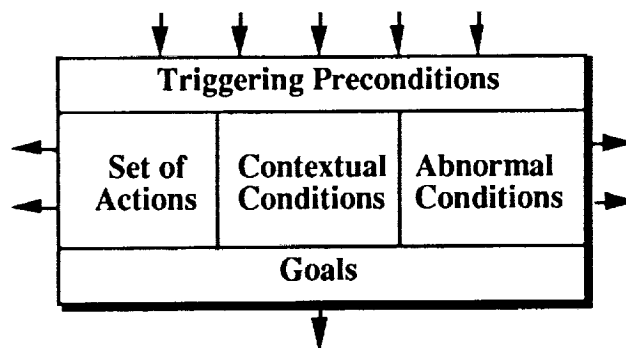


Figure 3.1. Block representation

Generally¹³, the selection of one (or several) action(s) in a *set of actions* is task-driven. The application of an action is supposed to reach a given sub-set of atomic goals, and can be attached to a particular set of abnormal conditions.

Triggering preconditions are conditions which lead to the activation of a set of actions. The triggering preconditions PC(B) of a knowledge block B are represented as a conjunction of predicates. The triggering preconditions of a block can be totally satisfied (necessity), partially satisfied (possibility), or not satisfied.

¹³ The concept of block has been used by Nathalie Mathé in her thesis in the telerobotics domain. The use of blocks in the documentation domain is slightly different (see Mathé, 1990).

Goals $G(B)$ of a block B are represented as a disjunctive set of predicates. Each goal is associated with an action and represents the expected result of this action. Goals of a block can be suggested, not suggested, reached, or not reached. A goal can be suggested by another block, or internally, when the triggering preconditions of its block are satisfied. Otherwise, it is not suggested at all.

An *abnormal condition* is represented as a disjunction of atomic predicates. Each abnormal condition of a block can be satisfied or not satisfied. When a set of actions is no longer applicable to the current situation, the situation is said to be "abnormal" for this block and an abnormal condition corresponding to this situation must be attached to the block. Abnormal conditions can be associated with the entire block or with a specific set of actions. For instance, during the execution of an aircraft navigation procedure (i.e. a particular piece of documentation), a cabin depressurization in flight is an abnormal condition which leads to the application of a recovery procedure.

Contextual conditions are represented as a disjunctive set of predicates. Contextual conditions in which a block holds, define a range for the state of the environment in which the block can be activated and executed. For instance, contextual conditions can be a set of conditions characterizing a failure of a particular device, the current goal of users, or a mixture of several physical and intentional conditions.

For instance, if somebody is solving a problem in a given environment, there are a lot of tacit preconditions which are "obvious" (e.g., constant) in this environment. These preconditions are included in the contextual conditions. Given this definition, if a system is in a context, reasoning can be done on the set of blocks belonging to that context. Thus, triggering preconditions can be made much simpler, leading to faster pattern matching. Contextual conditions are organized in hierarchies. This facilitates the organization of the block knowledge base. For example, in aviation, a flight is generally organized into phases, which are decomposed into sub-phases and finally each sub-phase is described in the form of knowledge blocks that are documented in the form of procedures (Figure 3.2). In the context of a flight, there are several sub-contexts (before take-off, take-off, etc.). In each terminal context (e.g., take-off), a set of blocks has to be performed. If everything is "normal", the pilot accelerates the plane up to a decision speed called $V1$ after which he will not be able to stop the plane in case of a major incident. He executes the first block "Before $V1$." If no abnormal condition occurs, he executes the blocks "Before Rotation" and "Before $V2 + 10$ ". If an abnormal condition is satisfied during the execution of the block "Before $V1$ ", then he can decide to execute the block "Stop the plane".

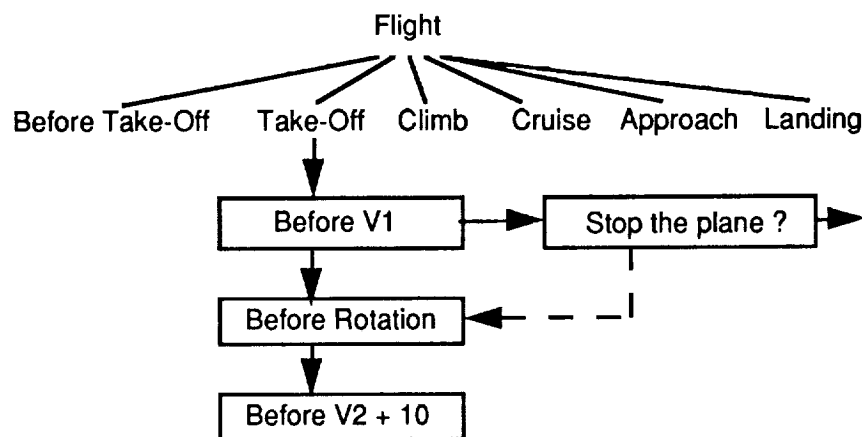


Figure 3.2. Simple hierarchy of contexts.

3.1.2. Context in Information Retrieval

3.1.2.1. External vs. internal context

In information retrieval, context may be seen as *external*, i.e., context is characterized by a set of attributes such as type of user, type of task leading to the information retrieval, etc., or *internal*, i.e., context is characterized by a set of attributes such as the location in the documentation where users are currently, the history of referents they have consulted so far, etc. The current context can be a set of external contextual conditions augmented by a set of internal contextual conditions. During any consultation of the documentation, the context may change either because users changed it, or because the current context has been changed from data coming from sensors, or because the location and history of the search changed.

3.1.2.2. Current context and context patterns

The *current context* is the expression of the situation of the current information retrieval (from both an internal and external standpoint). *Context patterns* express sets of contextual conditions that define a set of blocks in the same context.

3.1.2.3. Mutually-independent vs. dependent referents within a block

Within a given block, several cases are possible: proposed referents are mutually independent in the current context, or some (or all) of the referents can be grouped together in the current context (i.e., they must be displayed together in the current consultation either in sequence or in parallel).

If there is no information, referents within a block are considered mutually independent, i.e., they are connected by an exclusive disjunction operator.

It may happen that a subset of referents within a block are connected between each other i.e., they are connected by an conjunction operator. This conjunction operator can be specialized as a *procedure*:

- a sequence, i.e., these referents must be displayed one after another,
- a parallelism, i.e., these referents must be displayed all together at the same time.

Thus, a generic structure for referents in a knowledge block would be a disjunction of procedures (specialized conjunctions of referents), i.e.,

$$\bigvee_{j=1}^m \Delta_{r_i \in R_j} r_i$$

where m is the number of procedures, r_i is a referent, R_j is a subset of the set of referents for the procedure j , \vee is the disjunction operator, and Δ is the procedure operator (sequence or parallelism).

3.2. Using Contextual Knowledge

3.2.1. Block Management

When a context is recognized, the corresponding blocks are suggested. A block is executed when its triggering preconditions are totally or partially satisfied. Problem-specific criteria have to be introduced to define the term "partially satisfied". In the case of several partially satisfied triggering preconditions, the best precondition is chosen. The execution of a block consists of performing its actions and controlling the satisfaction of the corresponding abnormal conditions. Abnormal conditions can be of two types: weak abnormal conditions which, if they occur, will cause an exit from the current block towards another block in the same context, and strong abnormal conditions which, if they occur, will cause an exit from the current block towards another context of blocks.

For instance, a blown bulb in an aircraft cockpit may be a weak abnormal condition which does not necessitate changing the context of the flight. Conversely, an engine shut-down will cause a radical change of context, i.e., the pilot will adopt a different strategy (fly to the closest airport for example) and apply the appropriate recovery procedure.

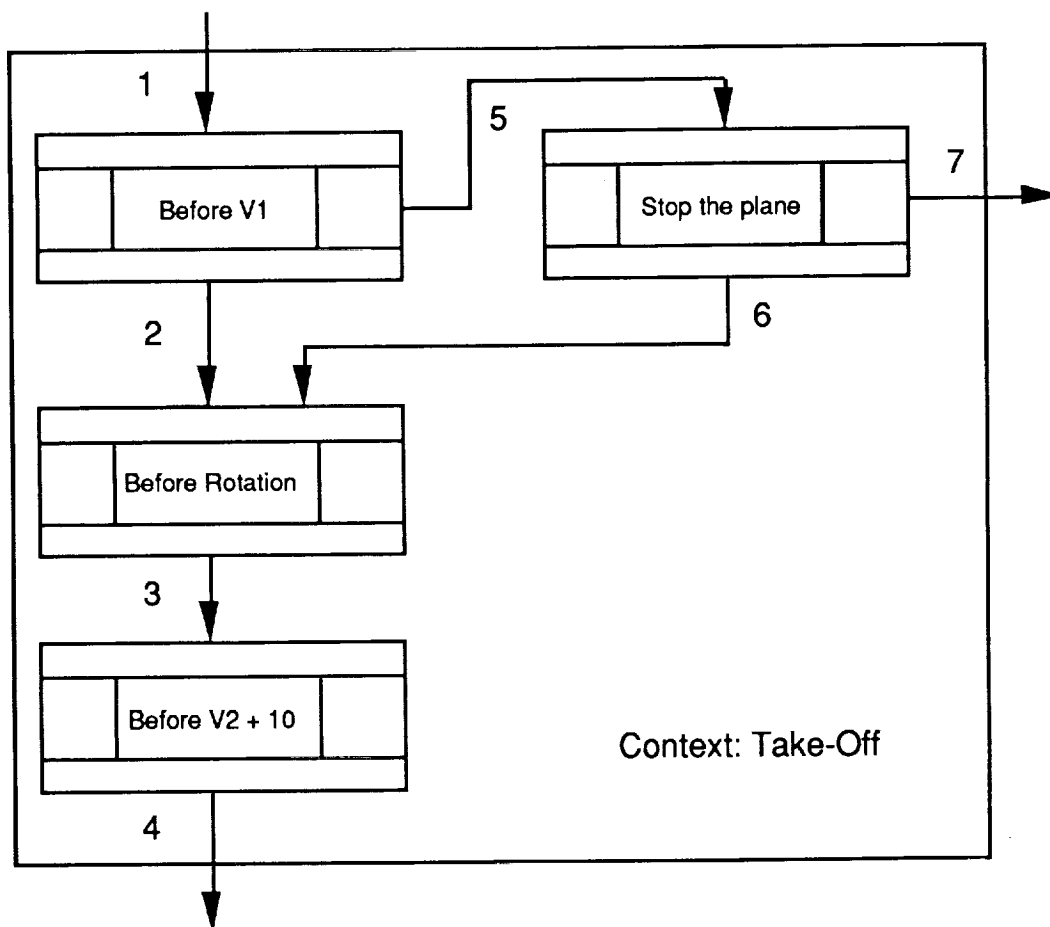


Figure 3.3. Context of knowledge blocks.

An example of a context of blocks is given in Figure 3.3 and represented in the form of a flow diagram. Let {"Before V1", "Before Rotation", "Before V2 + 10", "Stop the plane"} be a set of blocks having the same contextual conditions. Arrows represent links and are identified by numbers, from 1 to 7 in Figure 4. For example, link 5 links "Before V1" and "Stop the plane" when a specific abnormal condition of block "Before V1" is satisfied. Block "Before V1" is called the "root" of the context of blocks. Notice that a context of blocks is a block itself. In our example, triggering preconditions of the context block are the triggering preconditions of block "Before V1", its abnormal conditions are those of blocks "Stop the plane", and its goal is the goal of block "Before V2 + 10".

In the previous aviation example, each block has initially only one action that leads to the execution of a procedure. Thus, in this particular case, blocks and procedures correspond the same documentation entities. Blocks "Before V1", "Before Rotation", and "Before V2 + 10" stand for procedures "Before V1", "Before Rotation", and "Before V2+10". In normal situations, blocks are organized and processed in a tree sequence, e.g., "Before V1" -> "Before Rotation" -> "Before V2 + 10". We will call the resulting process *linear browsing* of a set of blocks. Abnormal situations interrupt this linear sequence to branch onto other blocks (generally called recovery procedures). In our aviation example, let us assume that, when applying the procedure "Before V1", the speed indicator shows an unacceptable value (weak abnormal condition 5). The pilot has to use procedure "Stop the plane" that monitors the thrust except if the speed is lower than a given threshold (strong abnormal condition 7). In this case, the pilot has to change the context and actually stop the plane. If the speed is within an appropriate range, then procedure "Stop the plane" succeeds and the pilot can apply procedure "Before Rotation" (link 6). We will call the resulting process *non-linear browsing*, e.g., A->D->B->C.

3.2.2. Using Context in Information Retrieval

Context is useful in information retrieval whenever there is more than one alternative referent (action) proposed to users when they select a descriptor (triggering precondition). As knowledge blocks are defined by context, each time a context pattern matches with the current context and the triggering condition is satisfied (typically, users select a descriptor), the corresponding knowledge block is activated and a set of referents is proposed to the user.

Use of context when there is a large number of blocks to consult at a given time is crucial. However, pattern matching may become very time-consuming in this case. Indeed, even if knowledge blocks increase the cognitive effect, i.e., the search is reduced, and more "cognitively" appropriate context patterns are built as a result of experience, the computational effect also increases, i.e., there are more context patterns to be scanned for a given current context and thus the run time is globally increased. Tambe and Newell (1988) have already shown this in the chunking mechanism of SOAR. Thus, replacing extensive search in the documentation base by an extensive pattern matching in a block base does not solve the problem of information retrieval, at least from a calculation time point of view. For this reason, context compilation may reduce the pattern matching.

3.2.2.1. Context compilation

The goal of context compilation is to reduce complexity. Context compilation is useful when there is a large number of context pattern candidates for a pattern match with the current context. Let EC the set of all the elementary conditions:

$$EC = \{c_1, c_2, \dots, c_N\}$$

where N is the number of elementary conditions.

Let C be the current context. C is characterized by a conjunction of contextual conditions:

$$C = (c_{\sigma_C(1)} \wedge c_{\sigma_C(2)} \wedge \dots \wedge c_{\sigma_C(n_C)})$$

where n_C is the number of elementary conditions in C , and $\sigma_C(.)$ is a function that maps the integer set $\{1, n_C\}$ onto the integer set $\{1, N\}$. Let B_j a block defined by the set $\{\pi_j, R_j\}$, where π_j is the context pattern in which B_j is valid, and R_j is the rest of the block description (i.e., triggering conditions, referents, and abnormal conditions). Let π_j a conjunction of contextual conditions:

$$\pi_j = (c_{\sigma_j(1)} \wedge c_{\sigma_j(2)} \wedge \dots \wedge c_{\sigma_j(n_j)})$$

where n_j is the number of elementary conditions in π_j , and $\sigma_j(.)$ is a function that maps the integer set $\{1, n_j\}$ onto the integer set $\{1, N\}$.

Complexity.

The complexity of pattern matching can be calculated as follows. If there are M block candidates for pattern matching, each block B_j includes n_j elementary conditions to be tested against n_C elementary conditions in the current context, the complexity can be expressed as:

$$(n_1 + n_2 + \dots + n_M) n_C$$

The complexity expressed as above cannot be greater than $M.N^2$. This complexity can be reduced by introducing heuristics in the pattern matching algorithm like stop testing elementary conditions whenever a context pattern fully matches the current context. However, in the current problem of information retrieval, we can reasonably expect that M will be much bigger than N and that the context patterns are highly connected, i.e., they generally share several elementary conditions. The main idea is to extract all the elementary conditions from all the context patterns and establish links between them and the corresponding context patterns. Let NEC the set of necessary elementary conditions to be tested to assure the pattern matching of the current block base:

$$NEC = \{c_{\sigma_{NEC}(1)}, c_{\sigma_{NEC}(2)}, \dots, c_{\sigma_{NEC}(n_{NEC})}\}$$

where n_{NEC} is the number of elementary conditions that are necessary to be tested to pattern match all the context patterns in the current block base, and $\sigma_{NEC}(.)$ is a function that maps the integer set $\{1, n_{NEC}\}$ onto the integer set $\{1, N\}$.

The context compilation we are proposing is based on a recursive construction of common subsets of context patterns, e.g., given π_j a context pattern having three elementary conditions:

$$\pi_j = (c_{\sigma_j(1)} \wedge c_{\sigma_j(2)} \wedge c_{\sigma_j(3)})$$

To be fully matched this pattern must have the following three elementary conditions matched also:

$$(c_{\sigma_j(1)}) (c_{\sigma_j(2)}) (c_{\sigma_j(3)})$$

Thus, the following compilation graph can be built (Figure 3.4):

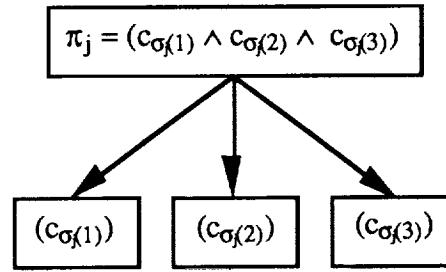


Figure 3.4. Compiled context structure for accessing the block base.

Using the resulting structure, the complexity is considerably reduced. The pattern matcher has to test only n_{NEC} elementary conditions.

3.2.2.2. Compiled context use

Given the current context C , a set of elementary contextual conditions NEC and a corresponding compiled context structure. The following algorithm can be applied (Figure 3.5):

1. Match C against NEC . The result of the pattern matching between C and NEC is the set:

$$MatchC = \{c_{\sigma_{MatchC}(1)}, c_{\sigma_{MatchC}(2)}, \dots, c_{\sigma_{MatchC}(n_{MatchC})}\}$$

where n_{MatchC} is the number of elementary conditions that are matched in the context C , and $\sigma_{MatchC}(\cdot)$ is a function that maps the integer set $\{1, n_{MatchC}\}$ onto the integer set $\{1, n_{NEC}\}$. Note that if complete pattern matching is required, $n_{MatchC} = n_C$.

2. For the first elementary condition in $MatchC$, prune all the context patterns in the current compiled context structure that are not connected to it, consider the resulting structure as the current structure, and delete the first elementary condition in $MatchC$.
3. If $MatchC$ is empty then the current structure is returned as the matched pattern(s), else return to step 2.

Figure 3.5. Compiled context use

3.2.2.3. Using abnormal conditions

As in the example described in section 3.2.1, abnormal conditions introduce non-linearities in information retrieval. In other words, referents are presented to users with the provision of information that they may not be satisfactory according to some attached abnormal conditions. This feature allows default reasoning.

3.3. Context-Sensitive Indexing

3.3.1. Building Suboptimal Block Bases by Semi-Automatic Indexing

Automatic indexing techniques have been described in sections 2.1.2, 2.1.3, and 2.1.4. We consider full-text indexing in this section.

3.3.1.1. *Extracting words and their frequency*

The first step extracts descriptors from text. We use a technique developed by Mark Zimmermann that allows full-text extraction of words associated with their frequency in the text. Zimmermann's software also allows browsing in internal context (i.e., in the context of the referent, see section 3.1.2.1) from these words in the original text. This capability can be used to build a first suboptimal descriptor base. The main output of this technique is a set of words with the number of their occurrence in the processed text, i.e.,

$$\{(d_1, n(d_1|D)), (d_2, n(d_2|D)), \dots, (d_p, n(d_p|D))\}$$

where p is the number of words extracted from the original text, $n(d_j|D)$ is the number of occurrences of d_j in the documentation D (see section 2.1.2.1).

3.3.1.2. *Extracting single-term descriptors*

From the set of words $\{d_1, d_2, \dots, d_p\}$ extracted by the techniques used in section 3.3.2.1, the goal is now to keep useful single-term descriptors that are included in this set. To do so, a stop list is used. It is also called the non-domain descriptor list. It includes all the single-term descriptors that are common function words. A previously acquired domain-descriptor list can be used to detect the new single-term descriptors. The internal-context-sensitive browsing capability provided by Zimmermann allows manual selection on this list of potential new single-term descriptors. The resulting set is:

$$\{(d_{\sigma_D(1)}, n(d_{\sigma_D(1)}|D)), (d_{\sigma_D(2)}, n(d_{\sigma_D(2)}|D)), \dots, (d_{\sigma_D(q)}, n(d_{\sigma_D(q)}|D))\}$$

where $q \leq p$ is the number of single-term descriptors, and $\sigma_D(\cdot)$ is a function that maps the integer set $\{1, q\}$ onto the integer set $\{1, p\}$.

3.3.1.3. *Constructing compound descriptors*

To our knowledge, there is no formal method readily available to construct compound descriptors from single-term descriptors. In this approach, we count on the motivation of the people involved in the indexing process to construct them from the resulting set of single-term descriptors provided by the technique described in section 3.3.2.2. This motivation can be improved if the analyst is provided with a good internal-context-sensitive browsing capability such as the one provided by Zimmermann. In other words, good editing capabilities have been observed to improve such a task.

Each time the analyst selects a single-term descriptor and its context, a list of lines is displayed. Each of these lines includes the corresponding single-term descriptor (in the middle) with a few words around it. If such viewpoint is not sufficient, the selection of the corresponding line causes the display of the original text in which this line is included. The

analyst may like to select several words around the single-term descriptor in this text and compose a compound descriptor.

3.3.1.4. Constructing aliases

As for compound descriptors, there is no formal method really available to construct aliases. Thus, good editing capabilities help users connect descriptors between each other. In CID, an alias mode is available that allows selection of several descriptors and connects them as aliases.

3.3.1.5. Building descriptor-referent links

Once a first list of descriptors is available, a full-text search can be performed and descriptors can be assigned to referents in which they are included. In our hypertext system, referents are labelled by the name of their support card. Full-text search is then performed sequentially in each card of the hypertext, and the block base is maintained incrementally along this search.

3.3.2. Intentional vs. Experimental Search

3.3.2.1. Definitions

At this point, we must introduce the two modes of activity performed in documentation use:

- *experimental search*: a casual approach, often seen in activities such as browsing and exploratory learning, which may imply a more active role for the computer in suggesting interesting information to be examined; note that experimental search will be also called *experimental browsing*;
- *intentional search*: a deliberate search for information to fill a particular need, for instance, to prepare a report or answer a specific diagnostic.

Experimental search is usually used to build initial relations between descriptors and referents that have not been built automatically using the technique described in 3.3.1. This can be done automatically assuming that descriptors are explicitly included in referents. In this case, our system scans the hypertext database and extracts each descriptor attached with a list of referents that corresponds to all the locations where this descriptor has been found. The corresponding blocks of knowledge that are generated this way are context-free. This automatic approach is generally not sufficient because some referents can be implicitly described by a descriptor, i.e., the descriptor is not explicitly written in the referent text. In this case, human intervention is necessary. We have implemented a mechanism that heuristically recognizes the experimental search mode when using the documentation. Thus, when a user acknowledges that a displayed referent is interesting (click on "Success"), the system asks for a description of this referent and subsequently generates a new block of knowledge in the context of the search.

Intentional search is used to refine existing blocks of knowledge, i.e., to add more context in them.

3.3.2.2. Detecting the user search mode

The user search mode can be set directly by users or inferred by the system using heuristics. Users can put the system in the experimental search mode (or the intentional search mode) by switching the search mode button on the control panel (see section 4.2.3.1). There are other ways to automatically turn the system into the experimental search mode. To do so, a few heuristics have to be defined such as:

- the user flips the pages,
- the user browses through the hierarchy of documents (nonlinear table of contents),
- the user browses the documentation in a full-text search mode, looking for a given descriptor in the text.

Note.

If users put the system in the experimental search mode by switching the search mode button, then the experimental mode persists, i.e., the system will not be able to switch automatically to the other mode. This remark introduces actually three effective search modes: free experimental, persistent experimental, and intentional.

We currently have only one heuristic defining the intentional search mode which is:

- the user selects a descriptor and consequently a referent (in the list attached to the descriptor) and did not put the system in the experimental search mode himself. This means that the selected referent is a goal which must be tested.

3.3.3. Success and Failure Feedback

Participation of the user to the indexing process has to be minimal and as non-invasive as possible. We have decided to provide the user with a success/failure set of buttons (each referent is equipped with this capability). Selection of the success (failure) button means that the user is (not) happy with the information provided in the current referent. Depending on the search mode, the system will behave differently (see section 3.3.4 and 3.3.5).

3.3.4. User-Guided Indexing in Experimental Search Mode

We distinguish user-guided indexing in experimental search mode (ES-mode) from semi-automatic indexing along the following dimensions:

	Semi-automatic indexing	User-guided indexing in ES-mode
documentation use type of indexing	off-line systematic	on-line casual

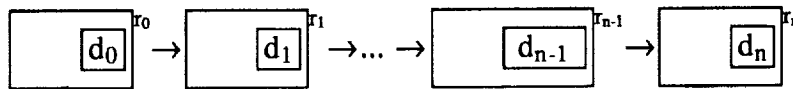
The first problem is for the system to recognize the user search mode. When the system knows that the user is in an experimental search mode, it must provide appropriate user-guided indexing capabilities.

3.3.4.1. Recording a trace

After the system detects the experimental search mode, it automatically starts to record descriptors and referents selected by users until they select the success button. The corresponding trace can be represented as follows:

$$(r_0, d_0) \Delta (r_1, d_1) \Delta \dots \Delta (r_n, d_n)$$

where for all i descriptor d_i is included in referent r_i , Δ is the sequence operator. When the user selects d_{i-1} in r_{i-1} , in the most general case, the system provides a set of possible referents attached to d_{i-1} , and then the user selects r_i among them. By definition, a trace is kept whenever the user selects the "success" button in the referent. In the above representation d_n (i.e., the last descriptor selected by the user) plays a special role, and will always describe the "success". In other words, when the user selects the button "Success", the virtual descriptor d_n is generated. The trace can be represented graphically as follows (see the representation of referents, section 4.2.1.2):



All stored traces are subsequently analyzed off-line (see section 5.3.3.1).

3.3.4.2. Describing a successful referent

Assuming that the system is in the ES-mode, when the user selects the success button, the system displays two lists of descriptors. The first list includes the suggested descriptors, i.e., the descriptors extracted from the trace. The second list includes all the possible descriptors. The user has to select one (or several) of them that describes the best the successful referent. If the corresponding block does not exist in the block base, then the system builds a knowledge block that includes the selected descriptor, the successful referent, and the current external context (although the user may modify the context if necessary).

3.3.5. User-Guided Indexing in Intentional Search Mode

IARC is the prototype system that implements the technique of Index Acquisition and Refinement according to Context that will be described in this section. The current version of the IARC system is implemented on a Macintosh II Cx in Allegro Common LISP, HyperTalk and C.

3.3.5.1. Discovering Abnormal Conditions

Following the same example as in section 1.3.4 of this technical memorandum, with the descriptor d_1 (triggering precondition) "air conditioning" and the contextual conditions (C_1, C_2, C_3) "you are a designer, you are concerned by the connection of the air conditioning system, and you have very little information about the electrical circuitry in the cabin", four different referents are possible in the documentation (r_1, r_2, r_3, r_4) "a list of the vendors of air conditioning systems", "a description of the air conditioning system", "a checklist of what to do when the air conditioning fails", "a diagram of the electrical circuitry in the main cabin of the Space Station". The first one is not satisfactory, it is a failure. The

second and the third are also failures. Fortunately, the fourth one will give the information that is needed, it is a success. If this result happens frequently, r_4 will be generated automatically from d_1 and (C_1, C_2, C_3) . Unfortunately, using this block of knowledge, a particular situation is observed in which r_4 is a failure. It would be unintelligent to repeat this experience again and again, which is why an abnormal condition will be added to the given block. In other words, what caused the goal not to be achieved? The resulting explanation will have to be added in the form of an abnormal condition.

```

(CONTEXT      (C1 C2 C3))
(TRIGG. COND. (d1)
(ACTIONS      (r1 +5 ((AC1 1) (AC2 3)))
               (r2 +3 ())
               (r3 +2 ((AC2 1) (AC3 1)))
               (r4 +2 ()))

```

Figure 3.6. Example of block.

Inputs to the program include user judgements on the success of actual retrievals. After a referent has been found, the user can select three buttons: "success", "failure", "don't care". The system automatically records this selection by adding +1, -1, or 0 to a reinforcement slot attached to the original block referent inferred by the descriptor used. An example of a block is presented in Figure 3.6. Descriptor d_1 is valid if the context conditions (C_1, C_2, C_3) are satisfied. The first referent r_1 has a reinforcement slot of +5; it has two abnormal conditions AC_1 with 1 occurrence and AC_2 with 3 occurrences.

The program's output is the set of blocks which are used as follows. When the user selects a descriptor, at any point in the documentation where this descriptor is, a menu of actions pops up with an ordered set of possible referents found successfully in the same context in past retrievals. The order of referents is based on the reinforcement slot. Such menus can be very different among users and in various contexts.

INPUTS: B, status (given by the user), and inputs from the user in case of failure

OUTPUTS: B, and a new block called newB in case of failure

METHOD:

```

IF status is success
THEN
  reinforce(currentAction of B, +1)
  IF currentContext  $\cup$  Context of B - currentContext  $\cap$  Context of B  $\neq \emptyset$ 
  THEN union(currentContext, Context of B)
ELSE
  IF status is failure
  THEN
    reinforce(currentAction of B, -1)
    elicitAbnormCond()
    put processAbnormCond() into currentAC of B
    incrementOccurrence(currentAC of B, +1)

```

Figure 3.7. Portion of the algorithm for the abnormal conditions acquisition

The following is a learning technique that augments system knowledge (Figure 3.7). The mechanism involves rote learning for incrementally capturing information about retrieval and adding it to the block contexts. The user gives information on the achievability of the goal of the action currently in use: *success* or *failure*. We will then say that the goal is

reached. This is done in a "given" current context c . If the current context c is included in the initial contextual conditions C_s of the block B , then a success may lead to a reinforcement of C_s . This is always the case if the user does not enter any description of the current context. However, if the user describes c , the following rules have to be considered. If the current context c and C_s have non-empty intersection, then success may lead to either a reinforcement or an extension of C_s . If the current context c is not included in C_s , then a success leads to an extension of C_s .

If the application of the action is a "failure", then an abnormal condition has to be inserted into the original set of abnormal conditions of the block. This abnormal condition is given by the user after the failure has been identified. Abnormal conditions can be seen as exceptions to the "normal" use of the block. In this case, the user returns to a manual operation and tries to provide a solution to recover from the failure. The trace of such a recovery is recorded and used for implementing a recovery block. A new block of knowledge is created and implicitly attached to the original one through an abnormal condition.

3.3.5.2. Refining Index-Blocks by Context

Using the above learning procedure, huge block bases can be generated. They have to be compiled to maintain reasonable memory size and performance. Contextual conditions are used to suggest a given context of blocks. At the beginning of the knowledge acquisition process, contextual conditions are very small, i.e., they include a few very simple conditions. As the process continues, parts of some blocks can be transferred into contextual conditions (also called situation patterns) (Boy, 1987). More generally, parts of some blocks can be transferred into other blocks upstream in the hierarchy. For instance, if a recovery block solving an abnormal condition is used often, then it cannot be qualified as a recovery block anymore and should be integrated (compiled) into the set of actions (referents) of the block(s) including this abnormal condition.

INPUTS: B , thresholdAC

OUTPUT: B , and eventually a set of newB

METHOD:

For all actions A in B do

IF exist AC_i associated to A such that $\text{occurrence}(AC_i) > \text{thresholdAC}$

THEN

Delete A and its associated abnormal conditions in B

Create newB having - the same context as B augmented by (not AC_i)

- the same descriptor as B

- the action A with a reinforcement slot of +1

Figure 3.8. Portion of the refinement procedure

The block refinement mechanism has been implemented as presented in Figure 3.8. Assume we are applying a block B described by a set of contextual conditions, a descriptor (or its aliases), a set of actions and their associated abnormal conditions. In the CID application, this block takes the form presented in Figure 3.6. The refinement procedure says that if the occurrence of an abnormal condition AC_i attached to an action A of a block B is greater than a given threshold then the corresponding action must be deleted in the block B and a new block must be created. This new block includes the contextual conditions of B augmented by the negation of AC_i such that when its descriptor (the same as B) is selected, it leads to the action A without abnormal condition attached. Of course, this new block could be augmented in future experimental or intentional search.

3.3.5.3. Example of Application

It takes years of training to become a flight controller in the Space Shuttle Mission Control Center. As one part of this training, people learn to use a large corpus of documentation to solve problems. They develop a deep knowledge of the organization of these manuals in order to access the proper sections as quickly as possible. Currently the operational documentation used by flight controllers is paper-based. For the short term, the goal of CID is to help people more efficiently access documentation on a computer. One thing CID attempts to do is to help narrow the search through documents while allowing the full browsing freedom to which people are accustomed. According to each user's situation, IARC will provide user-tailored assistance. CID will use IARC to incrementally learn new strategies of searching through documentation from observing people's use of documentation to solve problems.

One goal is to make CID look as close as possible to the documentation environment to which users are accustomed, the current organization of manuals in CID is unchanged and the pages are physically displayed the same way as in the manuals. As with regular books, users are allowed to write annotations attached to particular words or phrases they select. They can go through the manual sequentially or by using special facilities that hypertext systems allow. Generally, descriptors are used to go directly to a location in the manual where that descriptor is located. Users may also go directly to a particular section or page if they know where they want to go. They need not leaf through pages to find a certain section. This alone makes the process of searching through documentation easier.

When the user uses this tool and clicks on a descriptor, a menu pops up providing a list of possible locations for the desired information. The user may select a menu item and CID will provide the corresponding section to him. If the user is satisfied with this information, he may click on "success". This will reinforce the persistence of the selected item. If he is not satisfied, he may click on "failure". This will decrease the persistence of the selected item. Of course, each time the user cooperates (i.e. tells the computer to reinforce or penalize the persistence of an item), the corresponding context is taken into account by IARC which updates the corresponding block in CID. This way, IARC provides an intelligent assistant capability to CID. By acquiring more blocks and especially more context, CID should assist flight controllers in their search through documentation in an intelligent fashion.

In the example shown in Figure 3.9, the context was already known from CID and described by a set of strings such as Name_of_the_user (John Smith) and Type_of_task_leading_to_the_retrieval (Diagnosing). The user has clicked on the descriptor "OPS RECORDER", then CID provided the following menu (that is not displayed on Figure 5):

11.2. S-BAND_UHF LAUNCH REQUIREMENTS

11.46. LOSS OF RECORDERS

The user selected the item "11.2. S-BAND_UHF LAUNCH REQUIREMENTS". CID displayed the corresponding referent. The user did not like the corresponding information and clicked on "failure". Then CID provided the following menu:

11.46. LOSS OF RECORDERS

EXPLAIN WHY...

I AM LOST !!!

because the user may have wrongly selected the item "11.2. S-BAND_UHF LAUNCH REQUIREMENTS" (the system gives another chance to the user). If the user selects another location, the same process will take place. If the user can explain why the provided referent is not satisfactory, then he selects the "EXPLAIN WHY..." option. A list of explanations is provided to him by the system. These explanations are brief statements expressing abnormal conditions encountered in the past. He may select one of them or generate a new one. The selection is automatically processed and kept in the corresponding knowledge block as an abnormal condition. If the user selects "I AM LOST !!!", CID provides the opportunity to return to either the table of contents or the table of indices, or to change the description of the current context.

3.3.6. Conclusions

Context-sensitive indexing augments hypertext capabilities by encapsulating expertise that may be used to aid in the diagnosis, maintenance and repair of complex systems, choice of design alternatives, system configuration, intelligent tutoring, etc. It should translate users concerns into appropriate actions and help them solve problems. It allows adaptation of the system to users skill and knowledge.



10

10

Chapter 4

Implementation of a Computer Integrated Documentation System

The application to implement the first prototype of the Computer Integrated Documentation (CID) System is based on an analysis of Space Station Information Systems and our background on the problem of intelligent assistance (*Intelligent Assistant Systems* book by Boy, 1991). In this chapter, we report results of this analysis, and present the current prototype of CID.

4.1. Space Station Program Requirement Document Application

4.1.1. Introduction

Historically, NASA centers have developed their own information resources based upon the needs of each institution and limited to that individual site (Dede, Sullivan & Scace, 1988). Dede et al. claim that NASA documentation is characterized by its lack of integration.

There have been several initiatives reported by (Dede, Sullivan & Scace, 1988). The Technical Information Management System (TMIS) and the Software Support Environment (SSE) are critical information systems needed for the support of the Space Station program.

In a project such as the Space Station program, integrated documentation is crucial to connect all possible aspects of research, development, and operations. Everyone knows that documentation is generated painfully (even if a lot is generated!) and is rarely used in an efficient manner (when it is actually used).

The main problem in large documentation sets appears to be due to the lack of support for revision (documentation maintenance), and for contextual information retrieval (documentation management). This leads to inconsistent, incomplete, and redundant documentation systems. For instance, section labelling is generally inconsistent among documents¹⁴. Some documents have sections labelled: 1, 1.1, 1.1.1, etc. Other documents have: 1, 1.A, 1.A.1, etc.; or even: 1, A, 1, etc. Such inconsistencies generally cause problems in information retrieval.

¹⁴ A first answer to this problem would be to design a standard format that NASA writers would use. Taking into account that users have their own methods and styles, another solution would be leave generation of documents as is, and make a survey of the various ways people label sections of their documents. From this survey, it would be possible to build transformation rules from one style to another. Obviously, this approach assumes the fact that each writer structures his/her documents hierarchically.

4.1.2. Analysis of the Program Definition and Requirement Document

We have analyzed the Program Requirements Document (PRD) and the Program Definition and Requirement Document (PDRD). These documents have the same structure and include:

- a first page or cover,
- a revisions page that includes titles of the baseline issue and its reference number, revisions numbered by capital letters and their directive references number and their publication date (see section 2.2.2.1),
- a set of pages giving the paragraphs affected by the directive(s), i.e., a list of section numbers,
- a preface,
- a table of contents,
- a body of text (sections are numbered as 1.0, 1.1, 1.2, etc.), sections that have been changed are generally marked with a @ sign in the margin,
- abbreviations and acronyms,
- glossary or definitions,
- figures,
- tables.

It has to be mentioned that such documentation is changed very frequently during its lifetime. Thus, advanced capabilities of information retrieval and edition would improve its management and maintenance.

The PDRD is a detailed version of the Program Requirements Document (PRD). Both documents have to be maintained at the same time when revisions occur. That means that explicit connections would help in propagating the modifications in both documents. Furthermore, within a document an explicit connection structure would help retrieve information in context, e.g., one may need to know why and when such a piece of information has been generated in the PDRD. An explicit structure maintaining the history of changes would give this kind of information directly. At this stage, we are in the process of identifying deeper needs in this documentation management and maintenance.

4.1.3. The Technical and Management Information System

The Technical and Management Information System (TMIS) has been developed by Boeing for NASA (Figure 4.1).

The Program Automated Library System (PALS) provides an electronic documentation management capability that enables users to store, manage, track, and retrieve documentation, as well as full-text search capability for locating Space Station Freedom Program documentation. It includes: baselined documentation with text and graphics; in-process or working documentation with all generations; engineering graphics, drawings, and text; minutes; directives; change requests.

Such systems allow full-text search using equations of keywords. It also allows uploading and downloading of desired documents.

Problems with TMIS appear to be with its information retrieval mechanism based on basic keyword technology, i.e., it is not possible to include context-sensitive search in the documentation. Furthermore, the user interface would be very much improved by providing a hypertext-like capability using graphical maps and semantic indexing.

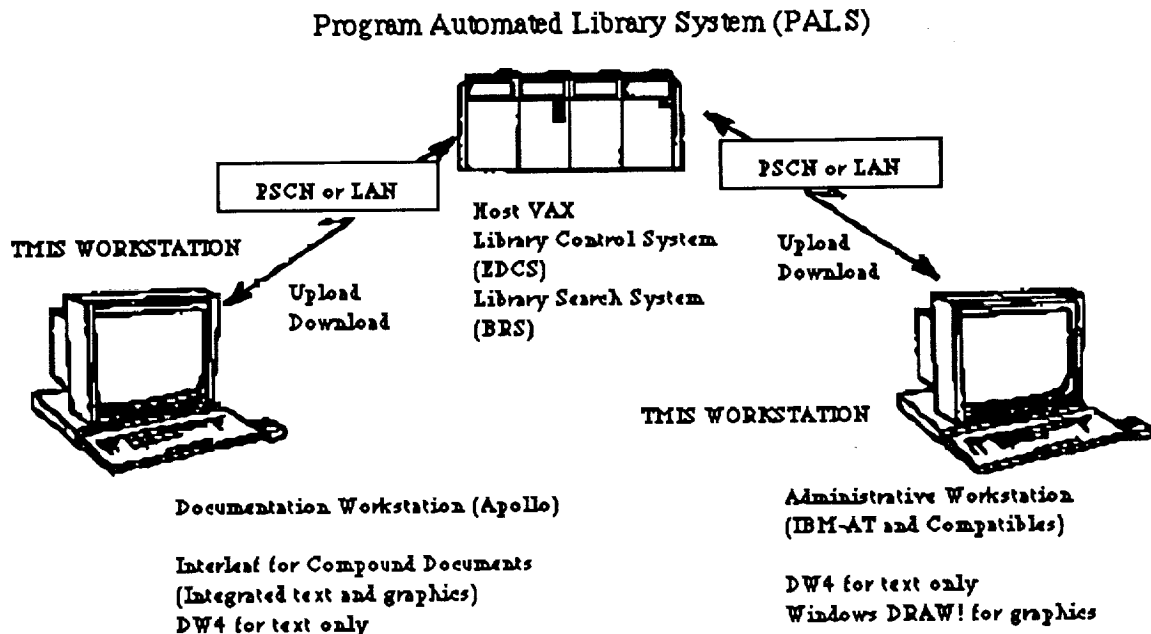


Figure 4.1. TMIS Document Management System.

4.2. The Computer Integrated Documentation System

The main idea in the infrastructure of CID is to split the knowledge useful for managing and maintaining documentation from the content of the documentation itself (i.e., text and graphics). Indeed, CID is composed of two main modules: the hypertext database management system and the knowledge-based management and maintenance system.

4.2.1. HyperText Database

The hypertext database includes a hierarchical structure of nodes (referents), and an interface management system that allows basic hypertext-like browsing in the documentation and interaction with the knowledge based management and maintenance system.

4.2.1.1. General infrastructure of referents

Documentation in CID is organized *hierarchically* (see section 2.2.2.1), i.e., there is a root document (in the Space Station documentation example, it is PRD). The other documents are hierarchically organized under the root, and documents themselves are structured hierarchically as explained in section 2.2.2.2. An example of hierarchy for a document is presented in Figure 4.2. A document is represented by a stack in HyperCard. Several documents can be open on the desktop at the same time (in the HyperCard 2.0 version). A consistent set of documents is generally called a collection. The basic entity is a card that includes three basic fields: hierarchy to access this card, text or graphics, and a table of contents for this card (see section B.2 in Appendix B).

Brown (1989) supports the fact that hierarchically organized hypertexts with a minimum of "goto-like" cross-references improves navigation. Taking this argument into account, we

have kept the hierarchical structure of the original text as a support for developing the hypertext infrastructure. Such a hierarchy constitutes permanent links between referents. In the hypertext, the cards containing the nodes are sequentially ordered depth-first. Links between cards can be *built-in* (e.g., go-to-next-card, go-to-index, etc.), or *content-dependent* (i.e., guided by a knowledge block). This distinction between links will be explained in the next section.

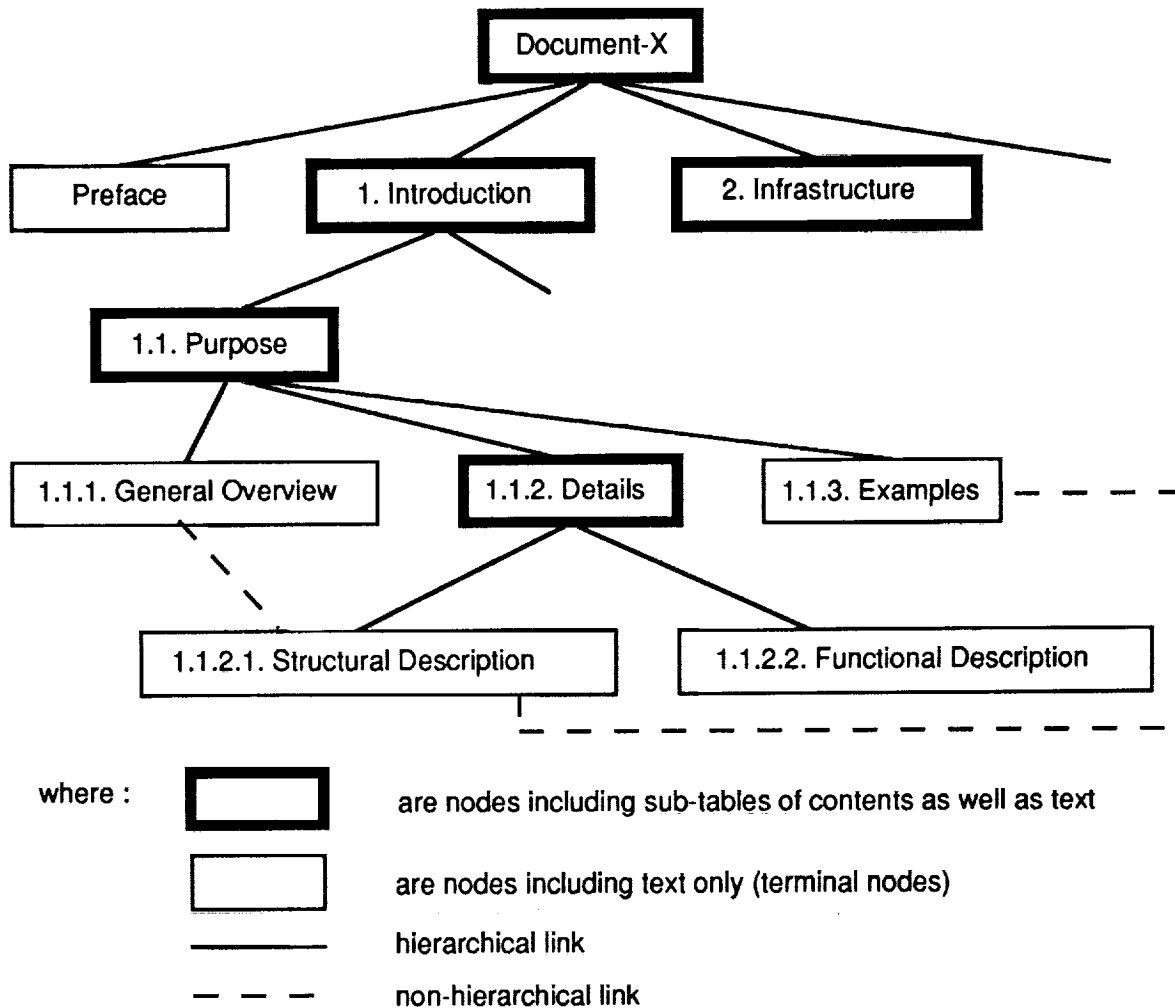


Figure 4.2. Example of CID hierarchy for a document.

4.2.1.2. Referents

Generic referent structure

A referent in the hypertext database is usually a card in a stack. It is characterized by:

- input-link descriptors,
- output-link descriptors, and
- internal behavioral variables.

Input-link descriptors are characterized by a set of descriptors $\{d\}_I$ that describe the referent and allow access to it. The descriptor that characterizes a referent without ambiguity is the section title (card name) and the document title (stack name), we call it $\{d\}_N$. The other input descriptors are words or phrases that can either be included in the text of the referent itself, we call them $\{d\}_T$, or describe the referent semantically without being explicitly present in the text of the referent, we call them $\{d\}_S$. The referent can also be displayed from built-in descriptors of other referents, such as next-card or previous-card, we call them $\{d\}_{BII}$. We have the global relation:

$$\{d\}_I = \{d\}_N + \{d\}_T + \{d\}_S + \{d\}_{BII}$$

Output-link descriptors are also characterized by a set of descriptors $\{d\}_O$ that are included in the referent and describe other referents. We have the same global relation:

$$\{d\}_O = \{d\}_H + \{d\}_C + \{d\}_{HIS} + \{d\}_T + \{d\}_S + \{d\}_{BIO}$$

where $\{d\}_H$ is a set of descriptors that allow automatic moves to a referent upstream in the hierarchy, $\{d\}_C$ is a set of descriptors that allow automatic moves to a referent downstream in the hierarchy (next level only), $\{d\}_{HIS}$ is a set of descriptors that allow automatic moves to a referent that has been already explored by the user (history), and $\{d\}_{BIO}$ is the set of built-in descriptors that allow exits from the referent and moves to another prespecified descriptor, e.g., back, next-card, index, general table of contents. A set of referents is attached to each descriptor included in the sets $\{d\}_T$ and $\{d\}_S$ that is presented to users when they select the descriptor. We have already seen that a descriptor and its attached referents constitutes a block of knowledge (without context for the time being).

Internal variables are characterized by a set of controls (buttons) and displays (fields of graphic bitmap) that allows management and maintenance of the referent, e.g., edit-text button. We say that these variables give a behavior to the referent. Figure 4.3 presents a systemic representation of a referent. Note that a referent can be activated from several descriptors, and several other referents can be activated from the set of output descriptors.

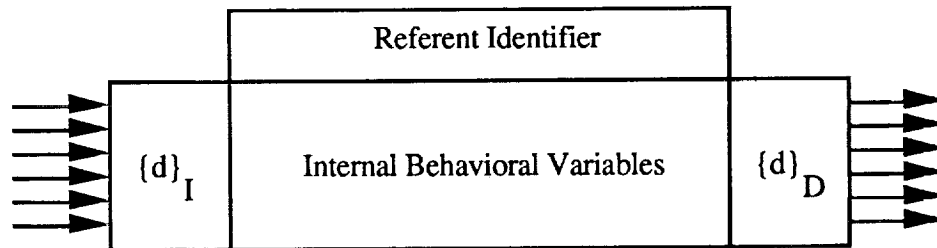


Figure 4.3. Systemic representation of a referent.

Single output assumption.

If we consider only one output possible at a time, browsing in the hypertext can be represented as a sequence of states described by descriptors and labeled by referents (Figure 4.4).

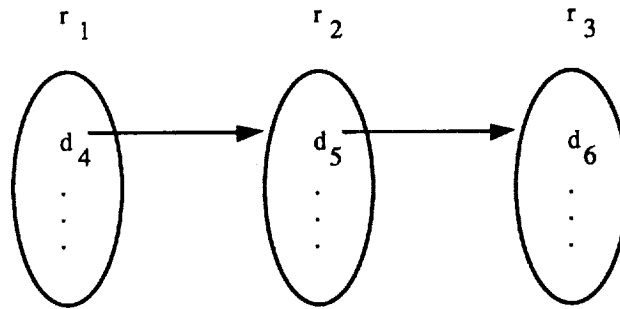


Figure 4.4. Sequence of referents as states of descriptors

4.2.1.3. Descriptors

Each referent includes a set of built-in descriptors and a set of programmable descriptors that are content-dependent.

Built-in descriptors.

Built-in descriptors (BID) are a set of predefined descriptors included in each referent of the documentation. They are currently the following:

- Next page (provides the next referent in the hierarchy in a depth-first search manner),
- Previous page (provides the previous referent in the hierarchy in a depth-first search manner),
- Back (provides the previously displayed referent),
- History (provides the list of the previous 20 referents displayed),
- Contents (provides the table of contents referent),
- Index (provides the index referent, i.e., alphabetical list of descriptors),
- Abbr. & Acr. (provides the abbreviations and acronyms referent),
- Definitions (provides the definitions referent),
- References (provides the references referent),
- Text (provides full-text search to find the referent including the first occurrence of a given descriptor).

Content-dependent descriptors.

These descriptors can be built using methods described in sections 3.3.1 and 3.3.4. They generally appear in the text $\{d\}_T$. Descriptors can be built by users from their observation of the referent contents $\{d\}_S$. These descriptors are not necessarily explicit in the text but can be pieces of graphics. The construction of graphical descriptors is described in section 4.2.3.3. Graphical descriptors are always in $\{d\}_S$.

4.2.2. Knowledge-Based Management and Maintenance System

4.2.2.1. Descriptor agenda

A descriptor agenda is maintained and available at any time, and is called the index. A single-term descriptor agenda is also available. Each single-term descriptor is included in one or several (compound) descriptors. Access to a compound descriptor (triggering condition of a block) from a single-term descriptor is described in section 4.2.2.2.

4.2.2.2. Access to a compound descriptor (triggering condition of a block) from a single-term descriptor

There are several ways to select a compound descriptor present in the text of a referent. One common way transforms the compound descriptor into a single string by replacing the spaces between the words (single-term descriptors) by another character. Usually people use underscores, e.g., "Automation and robotics" becomes "Automation_and_robotics". This special character may be invisible to the user using a special font. The advantage of this method is that when the user clicks on any of the words of the compound descriptor, the compound descriptor is immediately selected. The main disadvantage is that each time a new descriptor is added or modified, such a compilation has to be done in all the documents that include this compound descriptor. Furthermore, this method does not allow partial pattern matching, such as matching "Automation" with "Automation and robotics".

Our approach to this problem is the following. When the user selects a word w in the text of a referent, this word is recorded by the system. Each referent is equipped with its own set of descriptors $\{d\}_T$ (compound in general). The system tries to match w with each descriptor of $\{d\}_T$. If any exist, the system builds the list of descriptors including w , we call it $L(\{d\}_T)$. At this point, there are at least two possibilities: the *close-world assumption* possibility, and the *open-world assumption* possibility. They are implemented independently, and are provided exclusively from each in the system, i.e., if one is available, then the other is not.

The close-world assumption possibility assumes that when users select a word in the text, they know what they are doing, i.e., users recognize a compound descriptor. In this case, the system also records the words surrounding w (internal context), e.g., previous and next words, w_p and w_n , and tries to match each descriptor in $L(\{d\}_T)$ with a window of the text to include the selected word w . For instance, this window could be $(w_p w)$, $(w w_n)$ or $(w_p w w_n)$. If there is still an ambiguity, i.e. the number of resulting matched descriptors is greater than one, then more words surrounding w have to be taken into account, and so on.

The open-world assumption possibility does not commit on the fact that the user necessarily knows all the compound descriptors. It assumes however that the user and CID will cooperate to find the best compromise in the list of available descriptors (in the current referent). When users click on a word, it is generally because they recognize in this word a semantic description of what they are looking for. Thus, interactively the system proposes the entire list $L(\{d\}_T)$ to them. Then, they may select one of them. This method does burden users, and gives to them the possibility to select descriptors that may not be explicitly in the text they are using. For example, if the user clicks on the word "robotics", the system proposes a list of descriptors including "robotics" and "Automation and robotics" that the user may prefer. This method augments the number of clicks (two instead of one!). However, it is simpler, and provides more flexibility and potential to the user.

4.2.2.3. Representation of referents

Each referent (physically described in section 4.2.1.2) includes its own knowledge base. In this sense, a referent can be considered as an agent which has its own knowledge to advise the user in the next move from it. The knowledge base is composed of knowledge blocks. Blocks have been already described in section 3.3.5.1. An example of block is:

(CONTEXT	(C1 C2 C3))
(TRIGG. COND.	(d1)
(ACTIONS	(r1 +5 ((AC1 1) (AC2 3)))
	(r2 +3 ())
	(r3 +2 ((AC2 1) (AC3 1)))
	(r4 +2 ()))

User feedback for knowledge acquisition is handled through two success and failure buttons. The mechanism used for context acquisition is described in figures 4.5, 4.6 and 4.7 which must be understood in sequence.

Figure 4.5 shows that context is at the top of the hierarchy in the knowledge processing of a block. After the login, the system automatically provides a default list of contextual conditions, e.g., the corresponding default user profile. The user may add or remove some contextual conditions. If the system is plugged into the environment (in the case of the CID in the space shuttle for instance), some other systems may provide contextual conditions according to their own assessment of the situation. When the context is changed, the system automatically prunes the knowledge base(s) to remove and add relevant knowledge blocks.

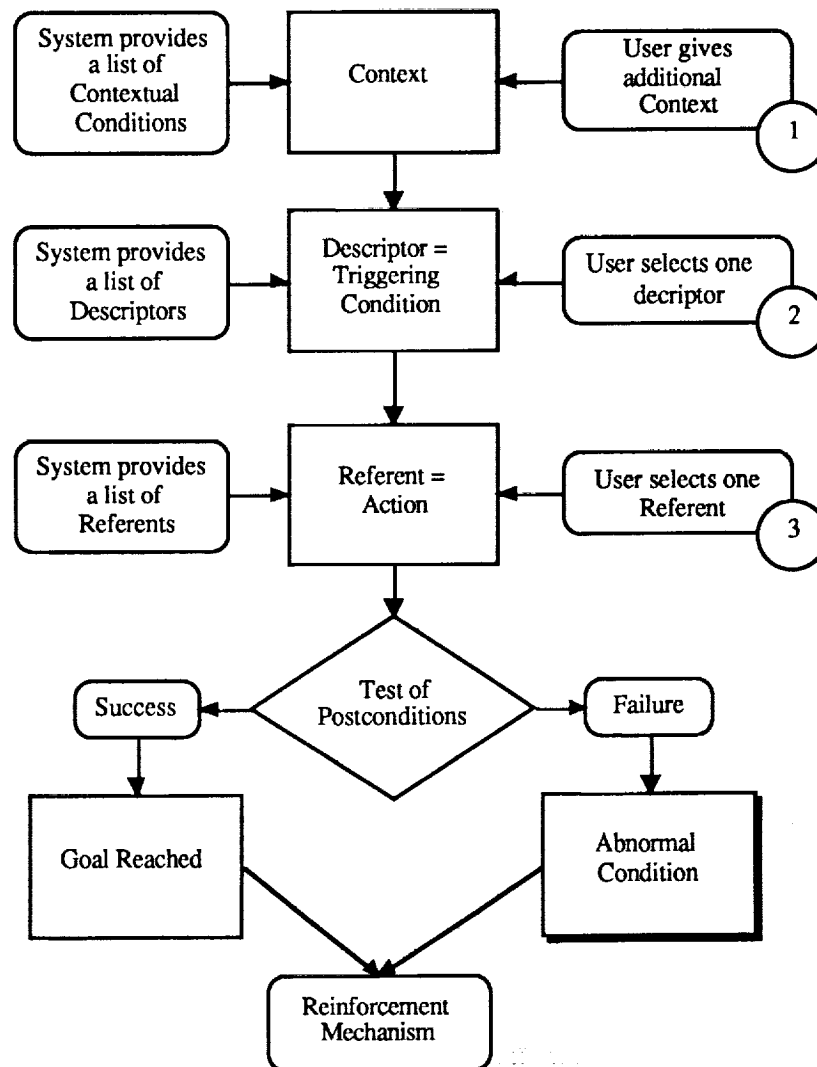


Figure 4.5. Intentional search and generation of user feedback.

A list of descriptors is either available in the text (or graphics) of the current referent, or in a special referent called the index. Whenever a descriptor is selected in a referent, the corresponding set of blocks in context is activated.

When a descriptor is activated, the system provides users with a list of referents (from the activated block[s]). They may select one of them (we will keep the single output assumption in this phase of the development of CID, see section 4.2.1.2). The corresponding referent is automatically displayed to users. Users then check if they are satisfied with it. If they are satisfied, then they select "success". In this case, the goal is said to be reached. If users are not satisfied, then they select "failure". In this case, the goal is not reached. This is typically an abnormal situation for the system.

Figure 4.6 presents various possibilities when an abnormal condition occurs. The first wrong cause might be that the user did not choose the right referent in the menu. If that was a bad selection (e.g., the user missed the right item in the menu), then the possibility of selection of another referent in the menu must be given. At this point, users have two possibilities, either they select another referent in the menu and go back to step 3 of Figure 4.5, or do not find any other interesting referent (i.e., opportunistically changes their mind) and start to browse the documentation (i.e., put them in an experimental browsing mode). If users think that they did not select a bad referent, then the choice of the descriptor must be questioned. If users did not choose the right descriptor, then they must have the possibility to select another one, either by going back to the previous referent (i.e., where the original choice was made), or by going to the index. In both case, users backtrack to step 2 of Figure 4.5. In contrast, if users persist on the choice of the descriptor, the context must be questioned. Users must have the possibility to update the context (go to step 1 of Figure 4.5), or eventually to switch into an experimental browsing mode.

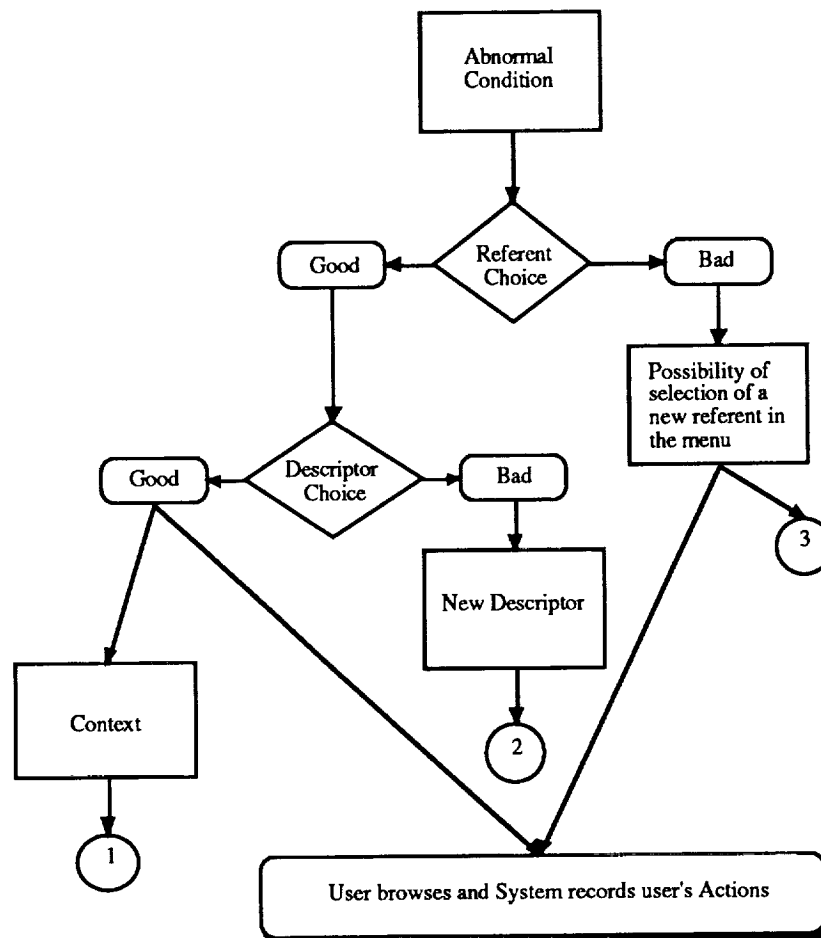


Figure 4.6. Various possible strategies after an abnormal condition.

Figure 4.7 presents repercussions of the experimental browsing mode on the block base. We have only developed the case of success, i.e., when the user is satisfied with a referent. The system then records the corresponding knowledge block. The system automatically asks the user to describe the successful referent. User are provided with an agenda of descriptors. They may select one or eventually add one to the existing list and select it. The system automatically builds a knowledge block with a reinforcement coefficient of 1. If it already exists, then the system just positively reinforces the link between the descriptor and the referent.

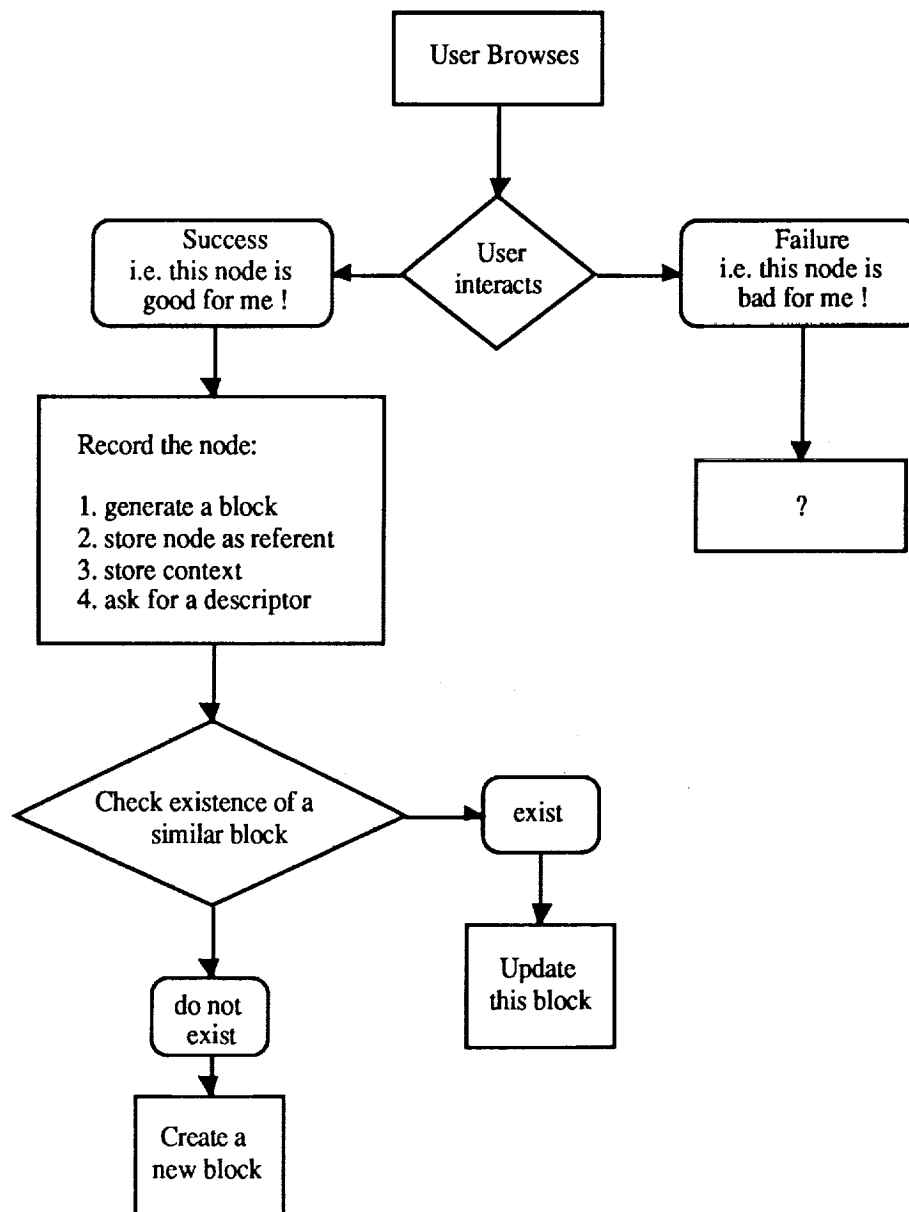


Figure 4.7. Experimental search and generation of a new block.

4.2.3. User Interface

4.2.3.1. Control panel

The CID control panel (CIDCP) is the central information management interface between CID and the user. As CID is a multiwindow system, the control panel is displayed in one of these windows. The others are used to display documents and other useful stacks, e.g., the agenda of document full descriptions. The CIDCP is composed of the following basic elements.

Displays.

CIDCP includes two main display fields:

- a. the main display field (MDF) used to present various kinds of information to the user, such as: the current descriptor, the number of open documents, the list of current open documents, etc.
- b. the listener field (LF) used by the CID to display messages to the user, or by the user to enter commands or natural language requests.

Controls.

CIDCP includes several control buttons:

- a. Context
- b. Current descriptor
- c. History
- d. Short description of a document
- e. Graphical navigation
- f. Open/close document
- g. Search/browsing mode
- h. Initial indexing (see section 3.3.1 for the description of the functionalities)
- i. Text-to-hypertext (see appendix B for the description of the functionalities)
- j. Hypertext-to-text

4.2.3.2. Basic stacks (windows)

The basic stacks are:

- a. Control panel window
- b. Context window
- c. History window
- d. Current open documents (text referent) windows (see sections 4.2.3.3 and 4.2.3.4)
- e. Abnormal conditions window
- f. Graphical navigation window
- g. Initial indexing window

4.2.3.3. Various kinds of referents

A referent can be a section, a document or a collection. A collection is physically represented by a stack of cards (in HyperCard™), each card of this stack describing a document of this collection (see Figure 4.8).

Collection Title	
Document Identification	Date
Document Title	
Author(s)	Directives
Preface / Revisions	
Controls	

Figure 4.8. Typical card describing a document.

A document is physically represented by a stack of cards (in HyperCard™), each card of this stack describing a section of this document (see Figure 4.9).

Document Title
1. Introduction 1.1. Purpose 1.1.2. Details
Text or Graphics
1.1.2.1. Functional aspects 1.1.2.2. Structural aspects
Controls

Figure 4.9. Typical card describing a section of a document.

4.2.3.3.1. Text referent

A text referent is represented by a generic card of the document stacks. This structure of such a card is described in appendix B section B.2. Additional capabilities such as editing are available, i.e., the user can edit the corresponding text¹⁵.

A text referent is composed of the following elements:

Displays.

- a. Document title field
- b. Scrollable field displaying the upstream hierarchy of the referent
- c. Scrollable field displaying the text expressing the content of the referent

Controls.

- a. Built-in descriptors described in section 4.2.1.2
- b. Content dependent descriptors that can be highlighted in the text to attract attention from the user. Other possibilities are under consideration such as superimposing a rectangle upon descriptors when the mouse spot is superimposed over them.

When the user selects a descriptor in the text, the system provides a menu of referents (Figure 4.10). Sets of dependent referents are separated by dashed lines in the menu.

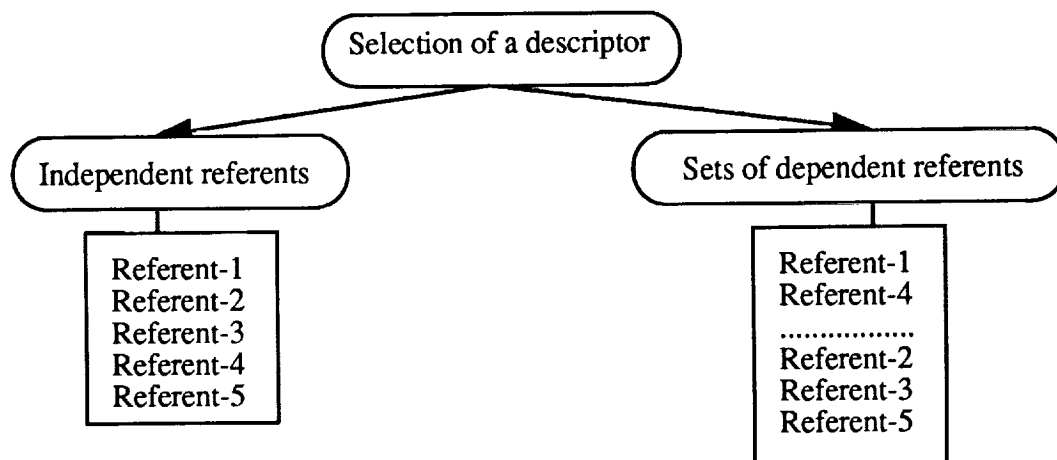


Figure 4.10. Menus of independent referents vs. menus of dependent referents.

Menu items (referents) are ranked by reinforcement coefficients weighted by the inverse of the number of descriptors in each referent. A first heuristic to detect dependent referents is to check if they are in the same hierarchy and they are part of a path in this hierarchy. In this case, the deepest one in the hierarchy is presented first.

4.2.3.3.2. Graphical referent

We have different support for graphical referents¹⁶. Graphics can be made using graphical software packages such as MacDraw, or scanned from already existing printed graphics. Such graphics can be pasted into cards, and then generate graphical referents. Our system

¹⁵ There is no concept of security in the CID system. This concept can be included as a context slot. Editing capabilities will be equipped with context-sensitive capabilities later.

¹⁶ This is mainly due to the capabilities offered by HyperCard.

allows the creation of mouse-sensitive zones that can be superimposed on such graphics. To generate graphical descriptors, it is sufficient to build mouse-sensitive zones and attach a text descriptor to them. The system provides such a capability.

4.2.3.4. Visual aids

As tests of visual aids is still a research issue, let's start to describe what they are. Visual aids are provided for users to augment their cognitive perception of the documentation content. These visual aids can be local or global and are generally graphical.

Local visual aids.

They are intended to give the user information about where to go next. If we take the road metaphor, this is equivalent to directions visible on the road signs as well as information available on indicators in the car. In other words, local visual aids corresponds to immediate information that the user needs to decide the next step in the documentation. Obviously, the corresponding decision making process should be decreased by providing the best possible information. This is the role of the knowledge blocks. Three kinds of local visual aids can be provided:

1. the current context;
2. menus of referents when the user selects a descriptor;
3. graphical connections to previous and next possible referents (Figure 4.11).

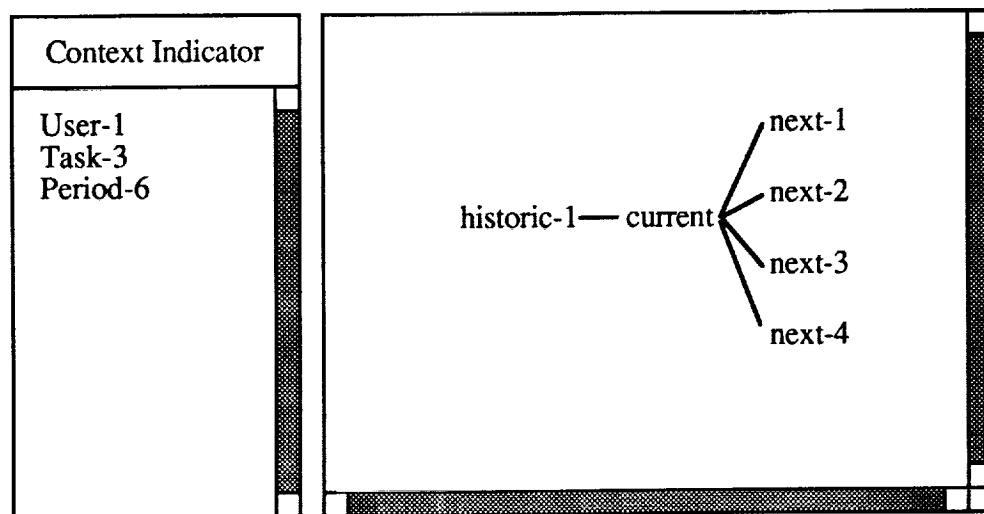


Figure 4.11. Local visual aids: current context and local graphical connections

Global visual aids.

They are intended to give users information about where they are now in broader perspective. If we take the road metaphor, this is equivalent to road maps. They can be (Figure 4.12):

1. graphs of nodes around the current node (a window on the documentation), these graphs can be either a hierarchical organization of the documentation or semantical relations between nodes (with possibility of zooming);

2. historical trace of what has been browsed so far showed either on a graphical map or in a list form.

Egan et al. (1989) compared people using a conventional book and a SuperBook (hypertext version of the original book). They found that readers used the table of contents (overview) much more in SuperBook than in the printed book and that they read about the same number of sections of the text even though they solved the problems in less time. This justifies the needs for global visual aids.

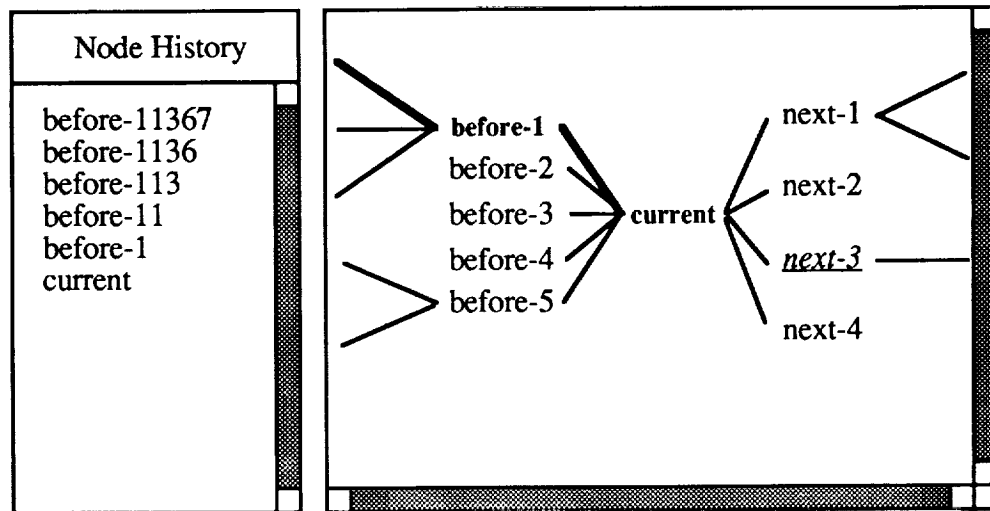


Figure 4.12. Global visual aids: node history and nodes around the current node where the node history is highlighted and the most relevant next node is suggested (here in italics underlined).

Foss (1988) claims that the browsing paradigm leads to two problems: the embedded digression problem of multiple sidetracks and redefinitions of current interests, leading users to forget the digressions they wanted to make, and the art museum phenomenon where you can spend all day in a large art museum and are not be able to remember any particular painting in detail. To alleviate these problems, this author has implemented four new kinds of browsing support in NoteCards where Foss did not believe that the original overview diagram mechanism was sufficient: graphic history lists, history trees, summary boxes, and summary trees.

Chapter 5

Theoretical Considerations

In this chapter we discuss some of the theoretical results and perspectives arising from our work, as well as the relation to other work.

5.1. Navigation in HyperSpace

The main basic assumption in this research is that the hyperspace is hierarchically structured. However, transverse links can be built. Two major definitions are needed. First, *indexing* is the process of building descriptors (descriptions) from referents¹⁷ and linking them together in a given context ($r \rightarrow d$). Second, *information retrieval* is the process of retrieving referents from available descriptors in a given context ($d \rightarrow r$). Information retrieval is a process of *abduction* using indexing knowledge, i.e., knowing $r \rightarrow d$ and d , r becomes a valid hypothesis. We say that indexing defines a set of semantic relations that are used in information retrieval. Figure 5.1 shows an example of semantic relations between referents via descriptors.

5.1.1. Theory of Navigation in Hyperspace

5.1.1.1. User's search mode and ontology

Automatic identification of users' search mode is still an issue. Several cases may occur:

- the user knows what they are looking for (they have seen the referent before). This is just a problem of recall,
- users do not know in advance the referent(s) they are looking for. This is a problem of:
 - reusing knowledge blocks that have been built before,
 - browsing and building their own indexing.

This overall process has been called: *index management and maintenance*. In other words, users browsing and formulating their descriptions of the referents content contribute to the augmentation of the block base. The more the block base grows, the more it becomes an ontology of the knowledge included in the documentation as perceived by users. The CID

¹⁷ Definitions of descriptors and referents are given in section 2.1.1.2.

can then be considered as a knowledge acquisition tool for users to build their own ontology of the documentation.

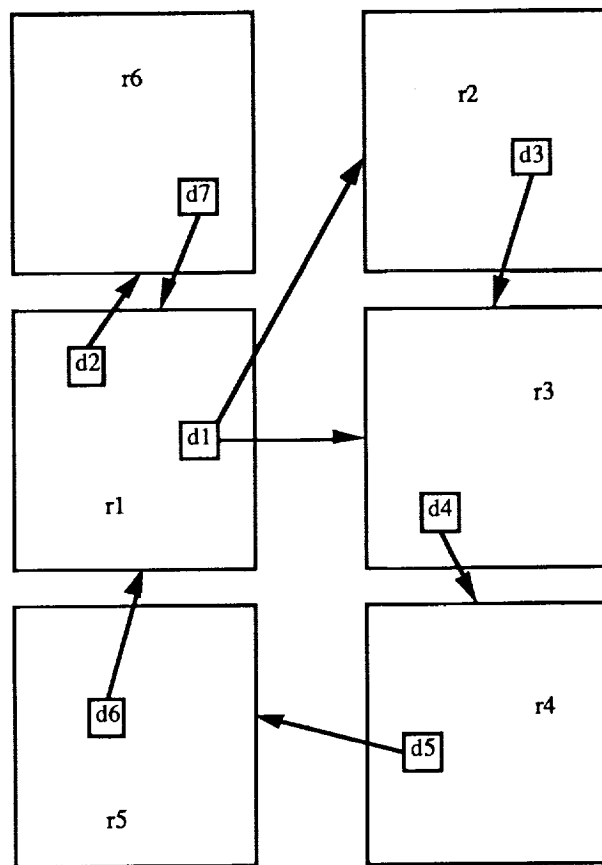


Figure 5.1. Example of semantic relations between referents via descriptors.

5.1.1.2. The block representation as a navigation aid

Knowledge blocks have been already described in Chapter 3.

Recursive aspect of blocks.

In a referent of a block, there may be a set of descriptors from which users can get another referent, and so on. This creates an implicit embedded block structure that can be represented as in Figure 5.2.

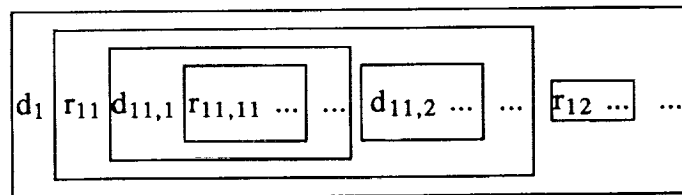


Figure 5.2. Embedded block structure.

where the block $(d_{11,1}, \{r_{11,11}, r_{11,12}, \dots\})$ is embedded in the block $(d_1, \{r_{11}, r_{12}, \dots\})$. In other words, when the user selects the descriptor d_1 and the referent r_{11} , if the descriptor $d_{11,1}$ is included in the referent r_{11} , then users have the opportunity to select in sequence the descriptor $d_{11,1}$ and the referent $r_{11,11}$, and so on. The figure 5.1 presents an example of semantic relations between referents via descriptors. Referent r_1 includes two descriptors d_1 and d_2 . The block $(d_3, \{r_3\})$ is embedded in the block $(d_1, \{r_2, r_3\})$. The path $(d_1, \{r_3\})$ is called a *short cut* of the path $(d_1, \{r_2\}) + (d_3, \{r_3\})$. Also, the block $(d_7, \{r_1\})$ is embedded in the block $(d_2, \{r_6\})$ and shows a *circularity*.

Consequently the block representation is capable of describing an integrated network of embedded descriptors/referents entities.

What to do next.

As we already mentioned, blocks can help users decide what to do next. The way descriptors and referents are organized defines the hyperspace topology. In other words, the infrastructure of the hyperspace influences the decisions of users when they browse. It is also reasonable to assume that users ability to find the information they are looking for in the hyperspace depends on their ability to navigate in such environment. This navigation problem includes knowledge about the navigation capabilities as well as knowledge about the domain (i.e., documentation contents). Indeed, it is clear that if users know about the vocabulary of the domain, descriptors-referents relationships are likely to be more understandable than for novices who have to learn and eventually build their own domain ontology.

An interesting research issue is to understand if navigation is content-sensitive or infrastructure-sensitive. This kind of research could be carried on by observing what users are selecting more often between the built-in descriptors and the content-dependent descriptors. Several issues may arise from this research, e.g., content-dependent navigation, hypertext infrastructure effect on user navigation, assessment of the internal structure of the documentation, and assessment of user's skill (comparison between novices and experts).

5.1.2. Hypertext Metalevel

Separating the knowledge-base indexing and information retrieval mechanism from the actual hypertext provides modularity of CID, i.e., both can be edited and maintained separately. However, this is not the only important consideration in this issue. Generally, indexing as well as information retrieval are performed by humans. This separation supports the concept of intelligent assistant system, as described in (Boy, 1991), built on top of the hypertext. It is indeed a metalevel on top of hypertext. A major advantage of this architecture is that this meta-level is easily programmable and allows the inclusion of knowledge on the links of the hypertext. The fact that the block representation maps well into the hypertext representation is a complementary reason to support such a separation.

5.2. Acquisition of Indexing Knowledge

5.2.1. Semantic Indexing

In the conventional approaches to indexing, we have seen indexing as a frequency-based process. Indexing would be improved if documentation of users' needs are taken into account in index generation and maintenance. Generally, users refine (and often define)

their needs by trial and error evaluating information retrieval results. Such an assumption implies the necessity of an interacting environment that allows incremental acquisition of indexing knowledge.

By semantic indexing, we mean building semantic relations between descriptors themselves, between referents themselves, and between descriptors and referents. Resulting semantic networks must be available to users in an easy-to-use fashion as described in section 5.1.1.2.

Indexing is a decision making process that must be directly accessible to the user. This decision process concerns building the semantic relations already described. The corresponding knowledge representation used to handle this decision process must allow repair in case of failure, i.e., when the user did not make the right decision. It should allow incremental transformation.

If semantic indexing has to be performed by users, how much can we ask them without disturbing them in their primary task, that is, in the usual case, information retrieval. The technique described in section 3.3 has been designed with this question in mind. That does not mean that all users will accept the CID semantic indexing job. This is an open question that still has to be tested. It is actually a basic research issue to investigate the interface requirements for intelligent human-machine cooperation. Acceptance of such new technology by users is a very complex issue that may involve long-term perspective. Indeed, experienced payback of this technique may greatly influence acceptance.

5.2.2. Extracting Blocks from Traces in the Hyperspace

5.2.2.1. Analysis of user's traces

In the experimental search mode, the system records users' actions, i.e., traces generated by users browsing in the hyperspace. When users select "success", this means that they are satisfied with the information they have gotten in trace T. However, it is not possible to know from this data if the complete trace is needed or part of it. If such a trace is recorded and analyzed off-line, it would be possible to know statistically what conjunction of referents the user needs when he selects a descriptor.

Analysis of the selected descriptors.

In the experimental search mode, the user selects descriptors that he finds close to his needs. However, all of them may not be relevant. The user may chose a descriptor because other potentially more relevant descriptors are not currently available. The first problem is then to identify what are the relevant descriptors in the trace.

Analysis of the selected referents.

Some referents may provide a piece of needed information but not be sufficient for the user. This may be a reason for him not to select the "success" button and select another descriptor in the corresponding referent.

Analysis of the trace.

An important question is: shall we keep the last link ($d_{n-1} \rightarrow r_n$), or the link ($d_0 \rightarrow r_n$), or an intermediate link ($d_j \rightarrow r_n$ ($0 \leq j \leq n-1$)), or a chain of referents (including r_n) after d_j ?

It is important to notice that these problems are studied experimentally.

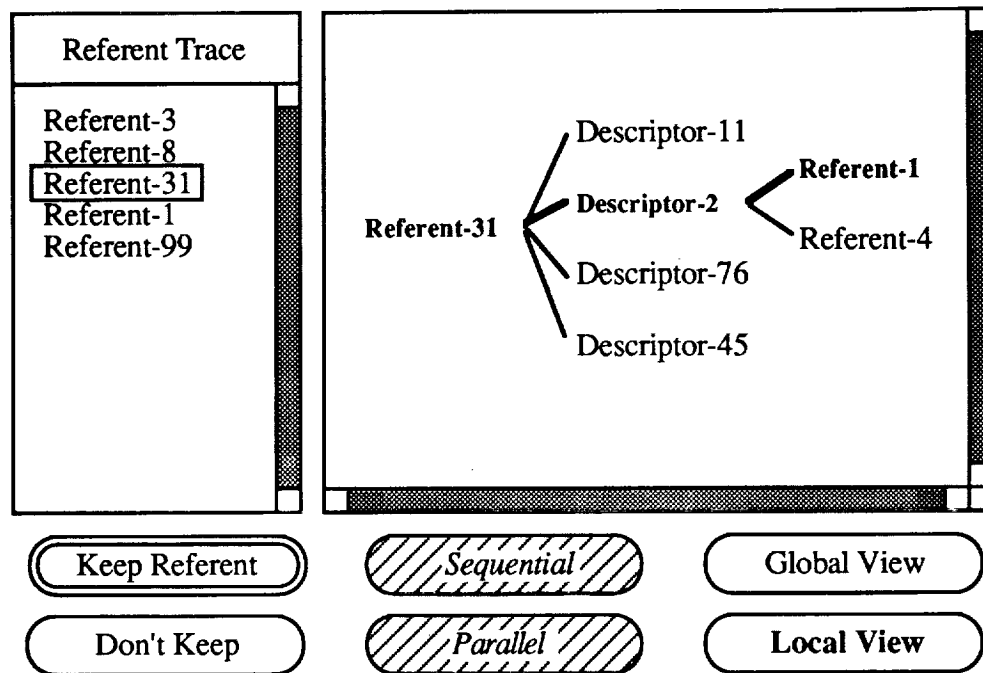


Figure 5.3. Interface for memorization of referents.

5.2.2.2. User interface capabilities for extracting useful referents-descriptors relations

We claim that it is more productive for users to analyze their traces immediately after the fact when they are still in context than off-line. This is due to the fact that users hardly remember what they did when they do not remember well the context of their actions. We already discussed the problem of user acceptance due to the secondary task imposed by this process of indexing in section 5.2.1.

To improve user ability and motivation to perform such trace analysis, an intelligent interface can be used (Figure 5.3). This interface should provide lists of descriptors, referents, and relations present in the trace in such a manner that the user can manipulate them easily and rapidly. We are still working on such an interface. Implementation and results will be available in the next report.

5.2.3. Context Clustering

We expect a large amount of blocks to be generated from the incremental knowledge acquisition. We also expect regularities coming out from the context patterns. For instance, the same class of users do the same things in similar situations. This can be handled by machine learning techniques such as learning from examples (Mitchell, 1982) or conceptual clustering (Fisher, 1987).

Our approach¹⁸ to construct conceptual clusters of contexts is incremental, i.e., new objects need to be assimilated one at a time. In his COBWEB system, Fisher (1987) proposes operators that allow incremental incorporation of new objects (in our case context

¹⁸ We have not started this particular effort yet. However, discussions have already started between the advanced interaction media group and machine learning specialists.

patterns) into a classification tree, where each node is a probabilistic concept that represents an object class. These operators include:

- classifying the object with respect to an existing class,
- creating a new class,
- combining two classes into a single class, and
- dividing a class into several classes.

We will take the same approach to context clustering. Let us assume that we have a block base including blocks of the form described in section 4.2.2.3. Each block includes a context pattern π_i such as defined in section 3.2.2.1:

$$\pi_j = (c_{\sigma_j(1)} \wedge c_{\sigma_j(2)} \wedge \dots \wedge c_{\sigma_j(n_j)}) \quad (5-1)$$

5.2.3.1. Placing a context pattern in an existing class

Quoting Fisher, placing a new context pattern in an existing class is probably the most natural way of updating a set of classes. Let's take an example that will illustrate the method. A new context pattern is (Paul Smith \wedge Diagnosing PGS). We already have the following classification tree:

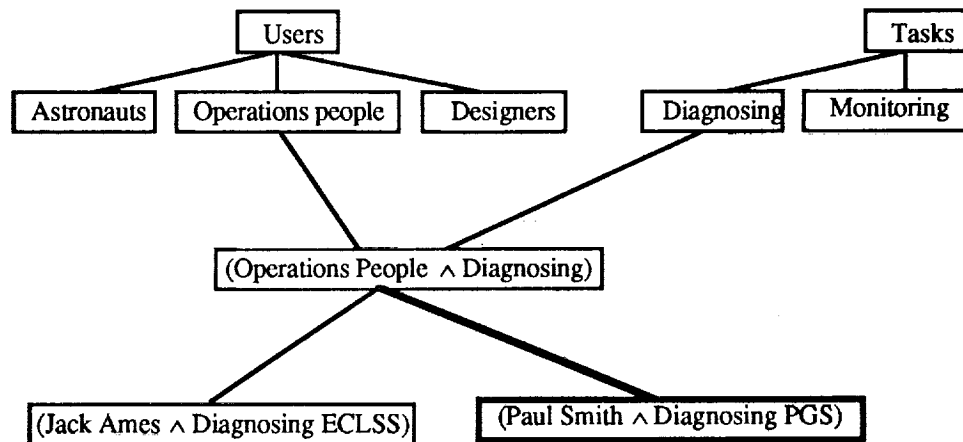


Figure 5.4. Placing a new context pattern in an existing class.

Each contextual condition $c_{\sigma_j(\alpha)}$ belongs to a class or is a class itself. In the current example (Figure 5.4), "Paul Smith" belongs to the class of "Operations People", and "Diagnosing PGS" belongs to the class "Diagnosing". Then the conjunction (Paul Smith \wedge Diagnosing PGS) is an instance of the existing class (Operations People \wedge Diagnosing).

5.2.3.2. Creating a new class

A new class is created whenever a new context pattern cannot be placed into an existing class. This new class is identical to the new context pattern. It can be linked upwards to

classes of some of the contextual conditions if any. In the current example (Figure 5.5), "Barry North" belongs to the class of "Designers", and "Diagnosing FTS" belongs to the class "Diagnosing". Then the conjunction (Barry North \wedge Diagnosing FTS) is an instance of the new class (Designers \wedge Diagnosing).

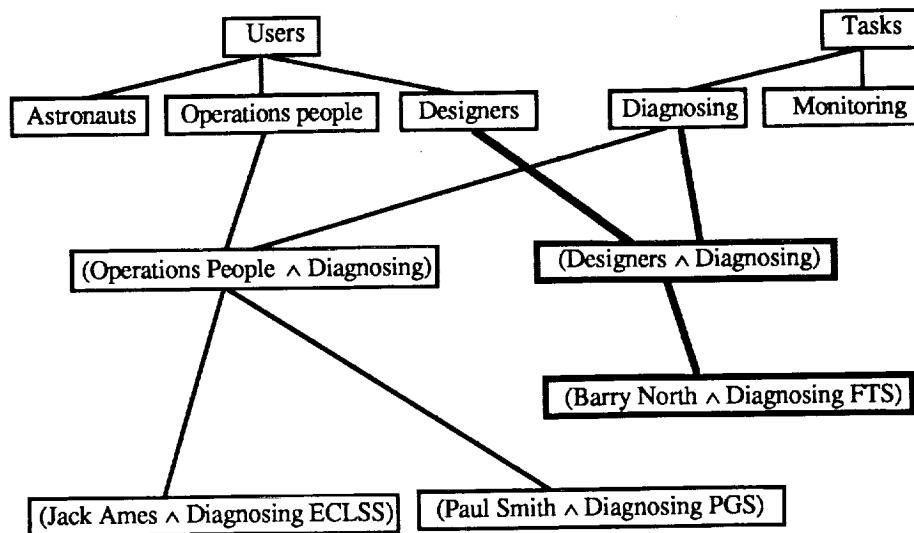


Figure 5.5. Creating a new class.

5.2.3.3. Merging several classes into a single class

As we assumed that compound classes are conjunctions of classes or objects, merging two classes involves creating a new class and replacing the instance-conjuncts by the corresponding class-conjuncts of the classes being merged. For instance in Figure 5.6a, "Operations People" and "Designers" are both "Users", and they are both performing a diagnostic task. Figure 5.6b shows the resulting context network after merging (Operations People \wedge Diagnosing) and (Designers \wedge Diagnosing FTS) into (Users \wedge Diagnosing).

The way we do merging can be called a generalization also.

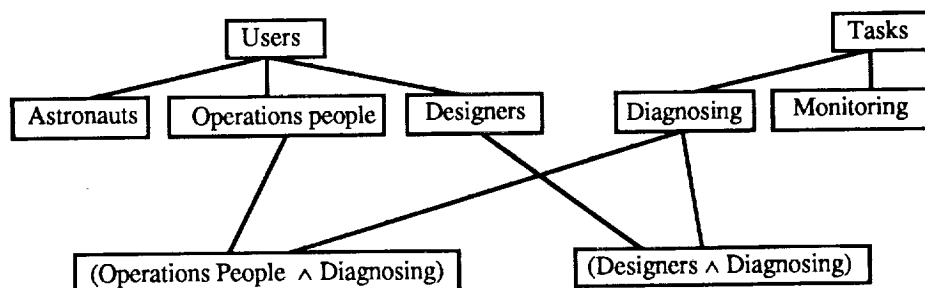


Figure 5.6a. Initial context classes before class merging.

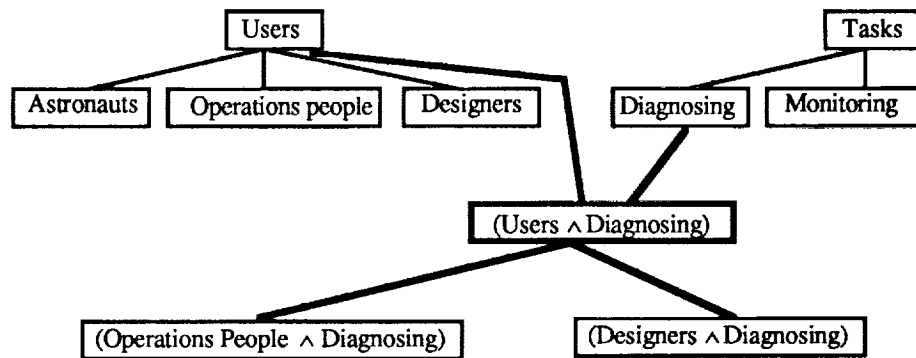


Figure 5.6b. Context classes after class merging.

5.2.3.4. Splitting a class into several classes

Splitting is roughly the inverse operator of merging (Fisher, 1987). Splitting happens when a compound class is an over-generalization. For instance, we may realize afterwards that $(Users \wedge Diagnosing)$ is too general because it happens that the class $(Astronauts \wedge Diagnosing)$ is very different from the other classes $(Operations People \wedge Diagnosing)$ and $(Designers \wedge Diagnosing)$. Thus, two classes will be created $(Astronauts \wedge Diagnosing)$ and $(Engineers \wedge Diagnosing)$ for instance (Figure 5.7). If the class "Engineers" does not exist already, this process leads to the creation of a new class, subclass of "Users" and superclass of "Operations People" and "Designers".

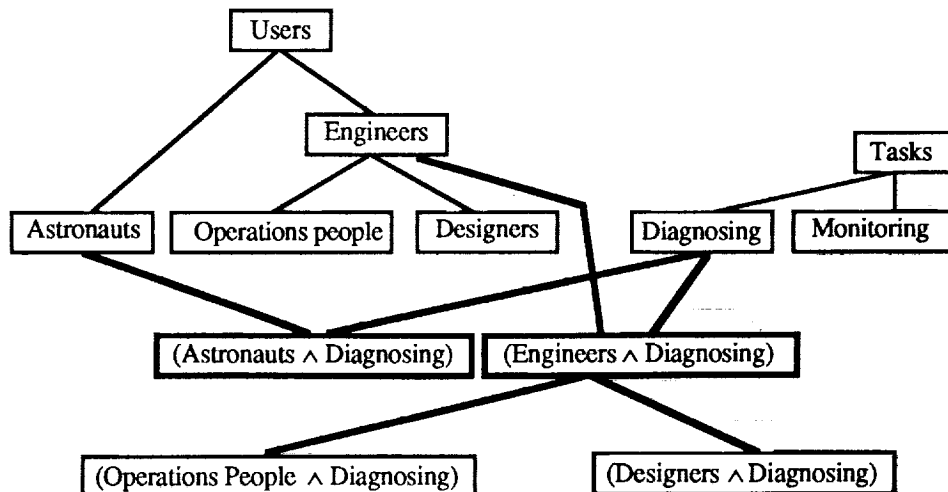


Figure 5.7. Splitting a class into several classes.

5.3. Generation and Maintenance of a Large Documentation

In the Space Station documentation for instance, documents are generated by individuals that are not necessarily connected between each other. They are theoretically connected through a top-down hierarchical structure. However, there is no transverse link anticipated by the current documentation system. This is actually a classical problem in the

construction of a very large documentation, e.g., sometimes two similar documents can be generated by independent teams. The problem is then to check if similar documents are describing the same things. If it is the case, can they be merged in a single document?

In order to solve this problem, it is necessary to generate descriptors that both represent documents they describe, and be eventually shared by other document generators if necessary. If these descriptors are well defined, we can expect that correlation between documents can be measured to check the degree of interdependence between documents.

5.3.1. Generation of Descriptors

5.3.1.1. Generation of descriptor at the user level

Documentation users should be provided with a dictionary of descriptors. This will reduce the number of new descriptors that have similar meaning as already defined ones (by other users). However, if users do not find any convenient descriptor in the dictionary, they must be able to define a new one. Definition of such new descriptors should be checked in the higher levels for consistency with the rest of the documentation. Thus, whenever a descriptor is generated, it must be sent up in the documentation hierarchy.

5.3.1.2. Maintenance of a descriptor dictionary

Documentation users should be provided with a thesaurus of descriptors. This will reduce the number of new descriptors that have meanings similar to existing descriptors (defined by other users). However, if users do not find any convenient descriptor in the dictionary, they must be able to define a new one. Definition of such new descriptors should be checked for consistency with the rest of the descriptors. In practice, whenever a descriptor is generated, it must be sent up in the documentation hierarchy. This consistency check problem is detailed below.

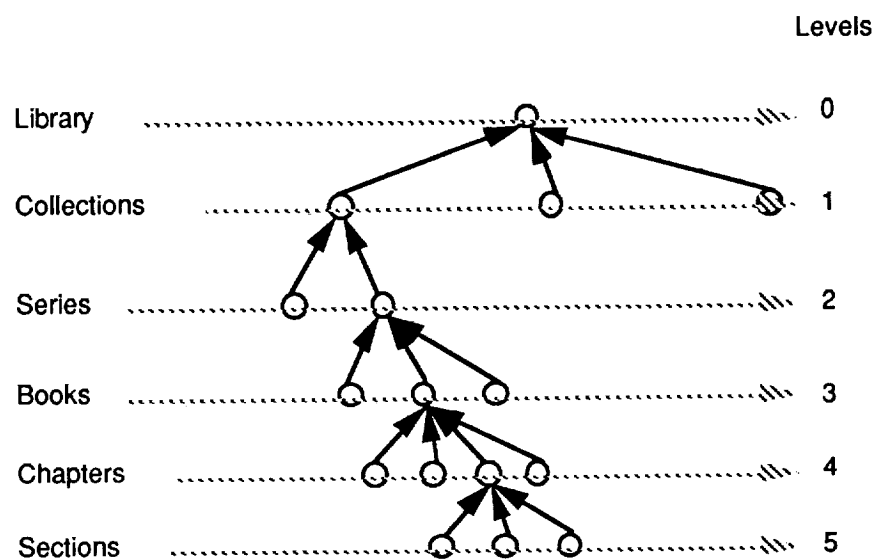


Figure 5.8. Example of hierarchical levels in documentation (document space).

There are two sets of levels that must be defined. First, a major assumption in CID is that the documentation is organized hierarchically. For instance, level 0 corresponds to the library level, level 1 to the collection level, and so on (see Figure 5.8). We will define these levels as hierarchical levels in the documentation or "doclev". Second, descriptors are also structured hierarchically. In this case, we will talk about hierarchical levels in the descriptors or "deslev". For instance, if a descriptor d_1 is defined at deslev $n-1$ and d_2 at deslev n , and d_1 and d_2 are dependent then d_1 is a generalization of d_2 . Even if doclevs do not map onto deslevs and conversely, there is some loose connections between them. The overall thesaurus of a documentation is ratified at the highest doclev. This ratification process also provides an explanation of why a new descriptor has been assigned to a specific referent. In particular, when a large documentation is being built, people designing higher doclevs must be consulted when new descriptors are being built in lower doclevs.

In CID, as each referent has an index (i.e., a list of descriptors), an index of a referent r at doclev n of the documentation structure should be a generalization of the index of a referent that is a child of r at doclev $n+1$ (Figure 2). The concept of inclusion has to be understood taking into account the concept of classes and objects. For instance, a descriptor at level n may be a class of another descriptor at level $n+1$. Let d_1 be a descriptor at doclev n , there exists necessarily a descriptor d_2 at each doclev $m < n$ that is either d_1 or a class of d_1 :

$$\begin{aligned} &\forall r_p, \text{doclev}(r_p)=n, \forall d_1 \in D(r_p), \forall m < n, \\ &\text{if } \exists r_q, \text{doclev}(r_q)=m \text{ and } r_p \text{ childOf}^{(1)} r_q \\ &\text{then } \exists d_2 \in D(r_q) \text{ and } d_1 \text{ childOf}^{(1)} d_2 \end{aligned} \quad (5-2)$$

where $\text{doclev}(r)$ is a function that maps the set of referents onto the integer set $\{0, 1, 2, \dots\}$, the relation " $r_p \text{ childOf}^{(1)} r_q$ " is true when r_p is a child of r_q in the referent space (the same definition applies in the descriptor space for " $d_1 \text{ childOf}^{(1)} d_2$ "), and $D(r)$ is the set of descriptors attached to the referent r . If the expression (5-2) is not verified for a particular descriptor d_1 then d_1 has to be added to (or a new class d_2 has to be defined that includes d_1) the index of referents r_q that are located on the path from r_p to the root referent (level 0 in the referent space). Thus, testing the satisfaction of (5-2) is a mean of improving completeness and consistency of the overall dictionary.

It is interesting to note that the descriptor space and the referent space should have the same partial ordering relations between related classes in the two spaces. In particular, if the descriptor space maps the referent space well, navigation in the documentation is facilitated.

5.3.2. Incremental Reinforcement from User's Feedback in Context

We have chosen the following heuristic to record a minimal set of parameters during the use of the system. We have noticed that the relevance of a contextual link $\{d_i, (r)\}$ between a descriptor d_i and a referent r depends on user's feedback (success or failure) on r , the frequency of feedbacks on r , and the importance that the user assigns to this referent r . The *semantic relevance* $\text{Rel}(r \mid d_i, C)$ of a referent r with respect to a descriptor d_i in context C has been formally defined as follows:

$$\text{Rel}(r \mid d_i, C) = \sum_{s \in S_{r \mid d_i, C}} \frac{I_s}{t - t_s} - \sum_{f \in F_{r \mid d_i, C}} \frac{I_f}{t - t_f} \quad (5-3)$$

where t is the current time, t_s (t_f) is the time when the user selected the success s (failure f), $S_{r \mid d_i, C}$ ($F_{r \mid d_i, C}$) is the set of successes (failures) so far at time t , and I_s (I_f) is the

importance assigned to r by the user with respect to d_i in context C . In practice, I_s and I_r may be set up as constants.

5.3.3. Semantic Correlation between Documents

In the Space Station documentation application, documents (referents) are generated by people that are not necessarily connected to each other. They are theoretically connected through a top-down hierarchical structure such as the one presented in Figure 5.8. However, there is no transverse link anticipated by the current documentation system. This is actually a classical problem in the construction of a very large documentation, e.g., sometimes two similar referents can be generated by independent teams. The problem is then to check if similar referents are describing the same things. If this is the case, can they be merged into a single referent?

In order to solve this problem, it is necessary to generate well-defined descriptors that can be shared by other documentation writers. If these descriptors are well defined, we can expect to check the degree of interdependence between referents by measuring their semantic correlation.

A semantic correlation between two referents should express the resemblance between the content of each referent. It can be constructed taking into account the descriptors attached to each referent. These descriptors characterize dimensions along which each referent can be located in the descriptor space. If descriptors express the semantics of the referent space, then the descriptor space can be called semantic space by extension. The main problem is that these dimensions are not independent. As already explained, the descriptor space represents relations between descriptors that express inheritance or property relations. Furthermore, it seems reasonable to let people index their referents without constraining them with the problem of descriptor dependence. The problem is then to design a representation that makes explicit descriptor dependencies. The basic idea is to build a *semantic network* where nodes are descriptors, and links are hierarchical (inheritance) or property links. Taking this approach, descriptors are organized into *descriptor dependency clusters*. These clusters may be loosely connected or actually independent.

We consider that the following parameters affect semantic correlation between two referents r_1 and r_2 :

- the number of descriptors $n_{ds}(r_1, r_2)$ shared between r_1 and r_2 ,
- the semantic relevance of a referent with respect to a descriptor in a given context (derived from user feedback),
- the hierarchical level of the descriptors in each referents,
- the number of properties shared by each shared descriptor¹⁹.

Let us assume that we want to compare two referents r_1 and r_2 . Salton (1989) defines the similarity between two referents r_1 and r_2 by a function of the number of descriptors shared by both referents. r_1 and r_2 have two descriptor lists $D(r_1)$ and $D(r_2)$. The intersection between $D(r_1)$ and $D(r_2)$ is computed using the following rules:

$$\begin{aligned} \text{If } d \in D(r_1) \text{ and } [\exists d' \in D(r_2), \exists n, \text{ such that } d' = \text{childOf}^{(n)}(d)] \\ \text{then } d \in D(r_1) \cap D(r_2) \end{aligned} \quad (5-4)$$

¹⁹ This parameter is expected to be relevant and important, however we do not take it into account at this stage of the development of CID.

$$\begin{aligned} &\text{If } d \in D(r_1) \text{ and } [\exists d' \in D(r_2), \exists n, \text{ such that } d = \text{childOf}^{(n)}(d')] \\ &\text{then } d' \in D(r_1) \cap D(r_2) \end{aligned} \quad (5-5)$$

where the function $d' = \text{childOf}^{(n)}(d)$ means that d' is a child of generation n of d . For instance, $\text{childOf}^{(0)}$ would be identity function, $\text{childOf}^{(1)}$ would be the direct child function (first generation), $\text{childOf}^{(2)}$ would be the grand child function (second generation), etc. In other words, we keep the most general descriptors in the intersection.

If the current context is taken into account, then the descriptor list must be restricted to the list of descriptors corresponding to the contextual links that are valid in the current context.

If the set $D(r_1) \cap D(r_2)$ is not empty, then it can be ordered with respect to their corresponding semantic relevances. We obtain the following table:

$D(r_1) \cap D(r_2)$	d_1	d_2	d_3	...
r_1	rank_{11}	rank_{12}	rank_{13}	...
r_2	rank_{21}	rank_{22}	rank_{23}	...

where rank_{12} is the rank of descriptor d_2 in the subset $D(r_1) \cap D(r_2)$ of the descriptors of r_1 . Each rank_{ji} is computed from the relative position of d_i in r_j with respect to the relevance $\text{Rel}(r_j \mid d_i, C)$. For instance, the above table shows the following order:

$$\begin{aligned} &\text{Rel}(r_1 \mid d_1, C) \leq \text{Rel}(r_1 \mid d_2, C) \leq \text{Rel}(r_1 \mid d_3, C) \leq \dots \\ &\text{Rel}(r_2 \mid d_1, C) \leq \text{Rel}(r_2 \mid d_2, C) \leq \text{Rel}(r_2 \mid d_3, C) \leq \dots \end{aligned}$$

The Spearman rank correlation coefficient applied to this problem gives the following formula (Snedecor, 1946):

$$\rho_{\text{Sem}}(r_1, r_2 \mid C) = 1 - \frac{6 \sum_{d \in D(r_1) \cap D(r_2)} [\text{rank}_{1,i} - \text{rank}_{2,i}]^2}{n_{\text{ds}}(r_1, r_2)^3 - n_{\text{ds}}(r_1, r_2)} \quad (5-6)$$

where $n_{\text{ds}}(r_1, r_2)$ is the number of descriptors shared between r_1 and r_2 , the following relation: $1 \leq \text{rank}_{ji} \leq n_{\text{ds}}(r_1, r_2)$ holds for $j=1$ or 2 , and $0 \leq \rho_{\text{Sem}}(r_1, r_2 \mid C) \leq 1$.

This semantic similarity measure can be used as a navigation aid. Two generic cases can be described as derivations of ordered lists of referents semantically correlated to: a given referent, or a complex query.

1. From a referent r_1 , $\rho_{\text{Sem}}(r_1, r_2 \mid C)$ may be computed for all the referents r_2 that share descriptors with r_1 .
2. A complex query involving the conjunction of several descriptors can be represented by a set $D(Q)$. An analogous formula $\rho_{\text{Sem}}(Q, r \mid C)$ can be derived for any referent r that shares at least one descriptor with Q .

In both cases, the corresponding ordered list of referents can be presented to the user as a suggestion of what to do next.

In the acquisition of contextual links, it can be used to fine tune referent descriptions. For instance, if two referents r_1 and r_2 are very much correlated but the user do not agree with this result, then referent descriptions $D(r_1)$ and/or $D(r_2)$ have to be revised.

5.4. Relation to Other Work

5.4.1. Regarding Blocks as Procedures

Index knowledge acquisition is related to procedural knowledge acquisition because of the common interest in the way actions are handled by users according to context. Acquisition of procedural knowledge from domain experts has been explored earlier in the MOLGEN system (Friedland, 1981).

A considerable effort has been already invested in modeling procedures. PRS (Procedural Reasoning System) work (Georgeff & Lansky, 1986) is very similar to ours. However, the notion of context in the knowledge areas (KA) of PRS is slightly different than the one used in the block KR. PRS does not currently have the notion of abnormal condition. Other work in this direction includes architectures which try to couple sensing to acting.

From the perspective of differentiating contextual conditions from triggering preconditions, similar ideas have been already developed in AGE (Nii & Aiello, 1979). In BB1 and GARDIAN, a similar representation has been implemented (Hayes-Roth, Washington, Hewett & Hewett, 1989).

From the perspective of representing insufficient or incomplete procedures, both humans and computers must be able to react promptly to new information, and they must be able to change or repair their knowledge when new information produces contradictions or when initial assumptions are withdrawn. Explicit representation of abnormal conditions or exceptions provides a simple mechanism for capturing knowledge when we discover it. Variable precision logic is the closest representational mechanism to the blocks (Michalski and Winston, 1986). Similar approaches have already been described in (Winston, 1983) and (Williamson, 1986).

The closest work has been done in telerobotics (Boy & Mathé, 1989; Mathé, 1990). In her thesis work, Nathalie Mathé has developed the block representation as a support for implementing intelligent assistant systems in process control tasks. Concepts that she has developed are very similar to those we have presented in this technical memorandum, even if she concentrated primarily on dynamic environments (by comparison documentation is a static environment). She did not develop any formal knowledge acquisition mechanism that could be implemented using blocks.

5.4.2. Intelligent Hypertext Perspective

To develop intelligent hypertexts, it is necessary to understand better how the users behave when they use such tools. In other words, work carried out on user modeling in this domain is very purposeful.

Canter, Rivers and Storrs (1985) define four graph theory-like classes of user navigation behavior: paths (a route that does not cross any node twice), rings (a route that returns to the node where it starts, this node being called the base node of the ring), loops (a ring that does not contain any ring as part of itself, i.e., it was a path until users returned to the base node), and spikes (a route where the return journey retraces (i.e., backtracks) exactly the route taken on the outwards journey). Based on these elementary structures, the authors characterize five different user navigation strategies: scanning (mixture of deep spikes and short loops), browsing (many large loops and few large rings), searching (ever-increasing

spikes with a few loops), exploring (many different paths), and wandering (many medium-sized rings). The authors compared users navigating a data set by hypertext and by direct command selection to desired nodes (a combination of goto and information retrieval) and found that hypertext users had many more rings and spikes than the direct access users but had about the same number of paths and loops.

Semantic indexing has been investigated by several authors. Dumais et al. (1988) propose a method for organizing nodes into a semantic structure on the basis of the overlap of the words used in those nodes. Stotts and Furuta (1988) proposed a model of hypertext based on Petri nets. Their system enforces browsing restrictions, e.g., deactivate some links. Weyer (1988) advocate the fact that information should be adaptable to the learner's preferences, and links should depend on the user's previous actions and current goals. This point of view support quite perfectly our knowledge-based approach to hypertext. A natural language (NL) approach to hypertext browsing has been proposed by Whalen and Patrick (1989). Their system interprets NL requests from the user in the context of the current location in the hypertext, and proposes new nodes according to this interpretation.

Fischer et al. (1989) propose a design environment integrating a hypertext system containing the rationale of the advice given by an AI system. The hypertext system uses an alternative implementation of the Issue-Based Information System (IBIS) method (Conklin & Begeman, 1988) to structure arguments for and against the various design options, and the AI system can then dump the user at the location in this hypertext that corresponds to the user's current undecided design problem.

Kibbyt and Mayes (1989), in their StrathTutor hypertext system try to eliminate the need for exclusively manual methods for creating links between hypertext nodes by generating links based on knowledge acquired when the user browses through the system.

Also, Monk (1987) presents a method for constructing a personal browser. In this approach, the system monitors the user's navigation behavior and interrupts the user to ask whether it should add a node to the browser when it has been accessed frequently.

Chapter 6

Personnel and Publications

In addition to the work described herein, this project included, as separate components, a study of procedure management and maintenance in the telerobotics domain, and a study of potential applications of intelligent assistant system in aerospace technology. For completeness, the personnel and publications listed below include all components of this on-going project.

6.1. Personnel

A number of researchers have been associated with this research effort. Guy Boy was the principal investigator. Jody Gevins, Bharathi Raghavan, Fabian Garcia Pastor (University of Madrid, Spain), and Joshua Rabinowitz provided a great deal of implementation. Nathalie Mathé helped with knowledge representation development. Thomas Gruber was associated as a Stanford consultant from the beginning, especially on the knowledge acquisition part. Ann Patterson-Hine and Joe Conley are applying CID to F-18 emergency procedures. Recently, Cécile Paris from the Information Sciences Institute of the University of Southern California, joined the project by adding a natural language processing contribution. Yaron Gold (NRC Research Associate) started to focus on the difficult problem of context representation and acquisition.

We also received a lot of feedback from the Boeing Advanced Technology Center in Seattle when they were involved in the Corporate Memory Facility, in particular Jeff Bradshaw and John Boose.

6.2. Major Publications and Presentations

Boy, G.A. (1989). Interactive Knowledge Acquisition for Intelligent Documentation. *Proceedings of the AAAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, MI, August 23.

Boy, G.A. (1989). The Block representation in knowledge acquisition for computer integrated documentation. *Proceedings of the Fourth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October 1-6.

Boy, G.A. (1989). Computer integrated documentation: a problem of knowledge acquisition and representation. *Proceedings of the ESA-ESTEC Workshop on Human Factors Engineering: A task Oriented Approach*, Noordwijk, The Netherlands, November 21-23.

Boy, G.A., & Gruber, T. (1990). Intelligent Assistant Systems: Support for Integrated Human-Machine Systems. *Proceedings of the AAAI Spring Symposium on Knowledge-Based Human Computer Communication*, Stanford, March 27-29.

Boy, G.A. (1990). Acquiring and Refining Procedure According to Context. *Proceedings of the AAAI-90 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Boston, MA, July 29.

Boy, G.A. (1990). Advanced Interaction Media. *Proceedings of the Third Conference on Human-Machine Interaction and Artificial Intelligence in Aeronautics and Space*, Toulouse, France, September 26-28.

Boy, G.A. (1990). Acquiring and Refining Indices According to Context. *Proceedings of the Fifth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 4-9. To appear in the Knowledge Acquisition Journal.

Boy, G.A. (1991). *Intelligent Assistant System*. Textbook. Published by Academic Press, London.

Boy, G.A. (1991). Presentation of a Poster on "Context Acquisition in Information Retrieval", at the Symposium on Learning Methods for Planning and Scheduling, held at Stanford University, January 5-6, 1991.

Boy, G.A. (1991). Computer Integrated Documentation. *MIT Conference on The Social Creation of Knowledge: Multimedia and Information Technologies in the University*, held at MIT, April 6, 1991. To appear in a MIT Press Book: The Social Creation of Knowledge.

Boy, G.A., (1991). Use and Acquisition of Contextual Knowledge in Information Retrieval. *Proceedings of the AAAI Workshop on Knowledge Acquisition: From Science to Technology to Tools*, Anaheim, California, July.

Boy, G.A. & Paris, C. (1991). An Intelligent Document Browsing System that Incorporates Indexing in Context. *Proceedings of the AAAI Workshop on Intelligent Multimedia Interfaces*, Anaheim, California, July.

Boy, G.A. (1991). On-line User Model Acquisition in Hypertext Documentation. *Proceedings of the IJCAI Workshop on Agent Modelling for Intelligent Interaction*, Sydney, Australia, August.

Boy, G.A. (1991). Some Theoretical Issues on the Computer Integrated Documentation Project. *Proceedings of the Sixth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October 6-11.

Boy, G.A. (1991). Indexing Hypertext Documents in Context. *Proceedings of the Hypertext'91 Conference*, San Antonio, Texas, December.

Appendix A

Measures of Vector Similarity

This appendix presents various measures of vector similarity introduced in section 2.1.5.1.2 of the text.

A.1. Inner Product

Evaluation for binary descriptor vectors:

$$|X \cap Y|$$

Evaluation for weighted descriptor vectors:

$$\sum_{i=1}^m x_i y_i$$

A.2. Dice Product

Evaluation for binary descriptor vectors:

$$2 \frac{|X \cap Y|}{|X| + |Y|}$$

Evaluation for weighted descriptor vectors:

$$\frac{2 \sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i^2 + \sum_{i=1}^m y_i^2}$$

A.3. Cosine Coefficient

Evaluation for binary descriptor vectors:

$$\frac{|X \cap Y|}{\sqrt{|X|} \cdot \sqrt{|Y|}}$$

Evaluation for weighted descriptor vectors:

$$\frac{\sum_{i=1}^m x_i y_i}{\sqrt{\sum_{i=1}^m x_i^2 \cdot \sum_{i=1}^m y_i^2}}$$

A.4. Jaccard Coefficient

Evaluation for binary descriptor vectors:

$$\frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

Evaluation for weighted descriptor vectors:

$$\frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i^2 + \sum_{i=1}^m y_i^2 - \sum_{i=1}^m x_i y_i}$$

Appendix B

From Text to HyperText

B.1. Introduction

Most of Space Station documentation is organized as described in section 2.2.3.1. Even if the text can be read linearly (i.e., page after page), it is also already organized hierarchically by documents, sections, subsections, etc. The main idea is to scan a document and extract the structure out of it. The corresponding hypertext version of the document will be an HyperCard stack organized in generic nodes such as presented in Figure B.1.

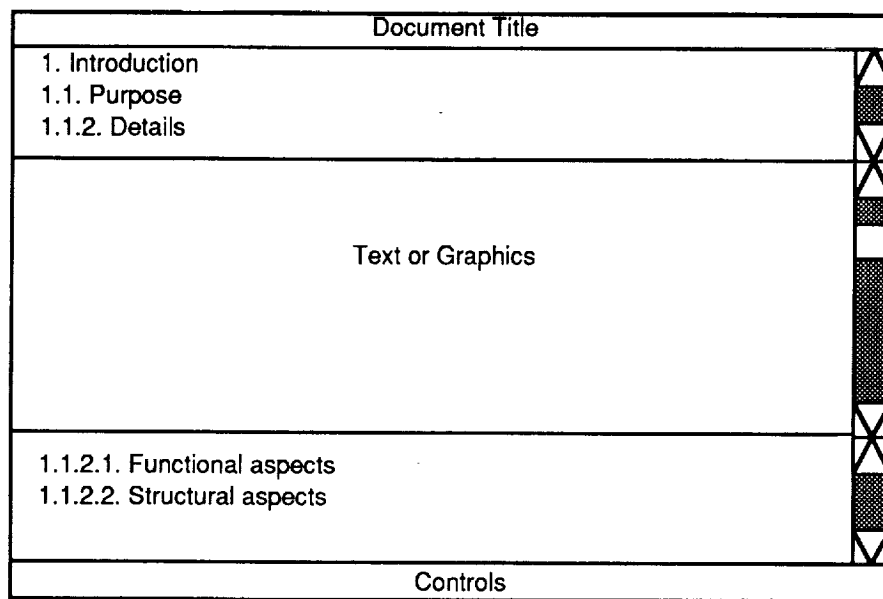


Figure B.1. Generic hypertext node (or referent) for the CID

A generic hypertext node or referent in the CID includes:

- a Document Title field;
- a scrollable field displaying the upstream hierarchy of the referent;

- a scrollable field displaying the text (or graphics²⁰) expressing the content of the referent;
- a scrollable field displaying the table of contents at the level of the current node, e.g., in figure B.1, the table of contents of the section "1.1.2. Details" is displayed;
- a zone including the controls described in section 4.2.3.3.

B.2. Text Referent Functionalities

B.2.1. Document Title Field

It displays the title of the document where the current referent is included. By clicking on it, the user will select the descriptor "First Page of the Document".

B.2.2. Hierarchy Field

It displays the upstream hierarchy of the current referent. Each line is a section title of the document. Higher levels in the hierarchy are displayed at the top. By clicking on a line of this field, the user will select the descriptor which will cause the display of the table of content of the corresponding section title.

B.2.3. Text Field

The text field display the content of a referent. Each time the user clicks on a word in the text, this word is selected and kept in a variable called "Holder" in HyperTalk. If this word is included in a descriptor and this descriptor is in a knowledge block, then the system displays a menu of referents attached to this descriptor. An algorithm for detecting if a word is included in a referent is presented in section 4.2.1.3.

As already mentioned in section B.2.2, this field can also be a table of contents. In this case, each line is clickable. By clicking on a line of this field, the user will select the descriptor which will cause the display of the corresponding section (that can be also a table of contents).

B.2.4. Built-in Descriptor Zone

Built-in descriptors are described in section 4.2.1.2. They actually include the "success" and "failure" buttons.

B.3. Construction of an Explicit Hierarchical Structure

We assume that the text to be converted in the CID format is written sequentially in the depth-first form of the hierarchical structure presented in figure 4.1. The goal of the following algorithm is to build an explicit hierarchical structure as shown on Figure 4.1 from a flat text. The source code of this algorithm can be obtain upon request.

²⁰ Note that this field is not yet scrollable for graphics.

-
1. Preprocess the original text file into a CID readable file.
 2. Compute all the hierarchy fields.
 3. Compute all the contents fields
 4. Create a new HyperCard stack and transfer text into this stack along with the hierarchy and contents fields.
-

For instance, from the text whose structure is described graphically in Figure 4.2 (section 4.2.1.1), the following sequence of cards is generated by the CID conversion mechanism:

Document-X, Preface, 1. Introduction, 1.1. Purpose, 1.1.1. General Overview, 1.1.2. Details, 1.1.2.1. Structural Description, 1.1.2.2. Functional Description, 1.1.3. Examples, ..., 2. Infrastructure, ...

B.4. Graphics Referent Functionalities

An additional function has been introduced that allows management and maintenance of sensitive graphic area (Figure B.2).

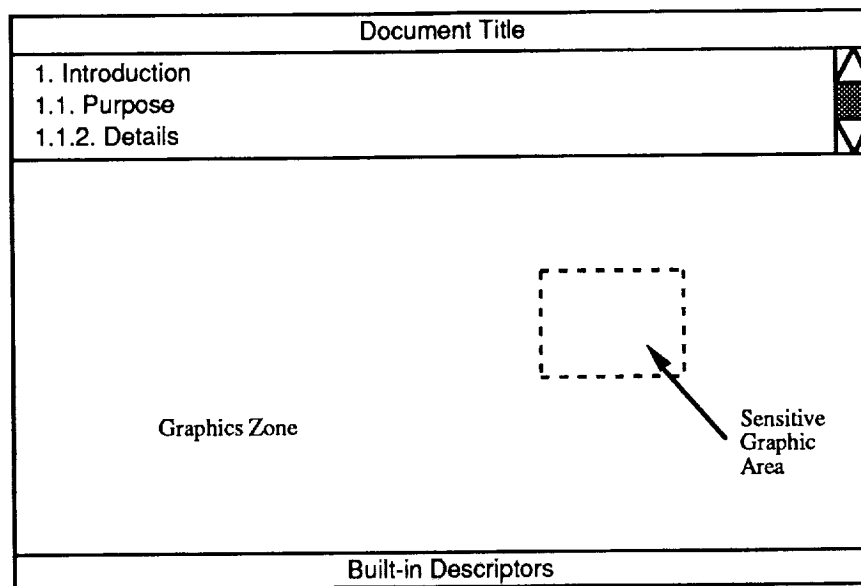


Figure B.2. Graphics node with a sensitive graphic area.

B.4.1. Sensitive Graphic Area Creation

The procedure to create a sensitive graphic area is the following:

-
1. Select the CREATE DESCRIPTOR button.
 2. System displays a dialog box asking for a descriptor name. Give a name and select OK. If CANCEL is selected in the dialog box, then no sensitive graphic area is created.
 3. Select an area on the screen (i.e., on the graphics) by clicking and dragging with the mouse. Doing this the sensitive graphic area is positioned and sized.
-

B.4.2. Sensitive Graphic Area Maintenance

Maintenance of a sensitive graphic area concerns modification of the descriptor name and the functions: resize and relocate.

Appendix C

Multimedia and Virtual Environments

C.1. Multimedia

Multimedia and virtual environments are exciting new technologies which appear to have great potential for intelligent assistance applications. We here provide a brief overview of the state of the art. *Multimedia* denotes the technology which provides connections between various information media such as text, graphics, images, sounds, voice, simulations, etc.

C.1.1. History

Multimedia technology has been made possible by improvements in storage technology (Fox, 1989). The first modern laser videodiscs were produced in the 1970s and had 54 000 analogue video frames (30 minutes of motion video) pressed onto each side of a 12-inch platter. The compact disc digital audio (CDDA) was launched in 1982, adapting videodisc technology for storing up to 72 minutes of high-quality digitally encoded audio on a 12-centimeter mass-produced compact disc. The first prototype of compact disc read-only memory (CD ROM) drives was demonstrated in 1984. CD ROM disks allow personal computers to access 600 megabytes of precoded digital data (e.g. text, pictures, databases). Writable optical disks (write once read-only: WORMs) became commercially available in 1984. WORMs supports are 12-inch platters. They are used for archiving a gigabyte or more of digital multimedia information. Compact disc interactive (CDI) specifications were announced in 1986. CDIs allow interaction with multimedia information. Digital video interactive (DVI) technology was first demonstrated in 1987 by a team of the David Sarnoff Research Centre. Seventy-two minutes of highly compressed full motion video and FM quality audio could be compressed in real time using special boards in an IBM PC/AT with attached CD ROM drive. Erasable optical discs became commercially available in 1988, storing hundreds of megabytes of multimedia information. In 1988, an ISO (International Organization for Standardization) standard specifying the volume and file organization of CD ROMs was approved, allowing inexpensive worldwide publication and distribution of multimedia data that are directly accessible on a wide range of hardware and operating systems.

C.1.2. Videodiscs

Video, like audio, consists of signals that pass through cables. There are three standards for analogue video in the world: NTSC in North America and Japan (30 frames per seconds [fps] and 525 lines per frame); PAL (25 fps and 625 lines per frame) in most of Europe; and SECAM in France and the USSR (25 fps and 625 lines per frame). Analogue video can be converted into digital video. An image, still or moving, can be converted into a video signal, most often through a video camera. This signal can be then recorded, most commonly on videotape. VHS and Video 8 are analogue composite consumer formats. Their picture quality and color reproduction are not intended to compete with professional and industrial formats (e.g. 1-inch "Type C", Component Beta). Video 8 is the newer and smaller format, and its quality appears to be at least as good as VHS. Both are highly accessible media, and many videodiscs have been produced in these formats.

Digital video requires more bandwidth than analogue video to produce the same result, unless sophisticated compression techniques are used. Digital disc recorders are like videotape recorders with large digital disc memories instead of tapes. They can hold 1500 – 3000 frames at a time, and can record and play back in single frame steps, at variable speed or at 30 fps. Digital disc recorders usually have two memory discs to allow internal recording playback. Videodiscs can be optical or magnetic, digital or analogue. Eight-inch and 12-inch laser discs are analogue composite read-only media, recorded once in a special production facility, and not erasable. CD-size laser discs have a variety of competing formats, including CDI, CDV (analogue with motion but short play), ICDV (analogue with motion but longer play) and DVI.

C.1.3. Multimedia Applications

Four types of applications are currently under development at MIT (Mackay & Davenport, 1989): interactive documentaries, learning environments, video data analysis and multimedia communication.

In the *interactive documentary* research area, film makers want to choose an optimal ordering of video segments. A good documentary tries to engage the viewer in an exploration. The film maker must develop a database of shots and edit lists, as well as an iconic representation of shots.

The goal of *learning environments* is to allow students to explore educational software. The MIT navigation project is a good example of the creation of multimedia object-oriented databases and learning environments. Hypermedia provides the user with a wider range of opportunities to explore, by following the links within a network of information nodes.

Human factors scientists use video recordings as data stores for analysis of human behaviour. They are more interested in the content of the recorded experiments than the format. Also, they need to mix other data, such as tracks of eye movements or keystroke logs, to make relevant analyses of experiments. An experimental video annotator (EVA) has been built by Wendy Mackay to help human factors scientists in *video data analysis*. It allows the scientist to create his own labels and annotation symbols prior to a session and permits on-line annotation of the video during a session.

Multimedia communication is an extension of the current media of long-distance communication such as electronic mail (asynchronous communication) and telephones (synchronous communication). The goal here is to provide a networked video editing system in order to share interactive video data. Users of on-line communication systems should be able to modify messages.

In multimedia applications a problem remains in the compression and decompression of video information. Applications that need real-time decompression still suffer from delays caused by various algorithms.

C.2. Virtual Environments

The main goal of virtual environments is to represent reality using appropriate technological media. This representation should take into account as closely as possible human cognitive and sensory capabilities to better capture and reproduce the look and feel of the real world it represents. A virtual environment can be defined as a multisensory simulation which can mix visual displays, sounds, relative motion, vibrations, chemical smells and wind simulation. The *Sensorama* project developed by Morton Heilig in 1962 is an example of an early system simulating a motorcycle ride through New York city (Lipton, 1964). In a virtual environment, the user is no longer a passive spectator like someone watching television. He is also an actor in the simulated environment, which reacts to his inputs. The user is sitting inside the image and sound. Dynamic aircraft simulators used for training pilots are an example of virtual environments. However, the term *virtual environments* has evolved and currently designates particular simulators where displays and other human-machine interaction devices have been mounted on the user himself, e.g. helmet-mounted display, the DataGlove and the DataSuit.

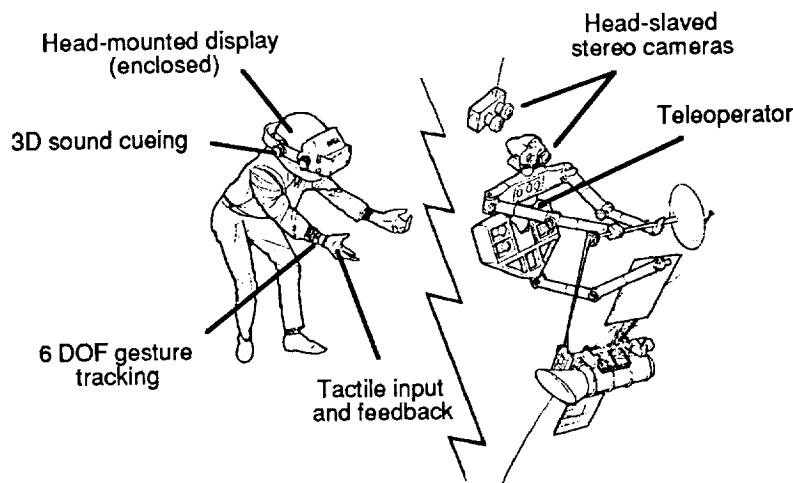


Figure C.1. Head-mounted display and DataGlove for virtual manipulation.

The *Aspen Movie Map* project, developed at MIT by the Architecture Machine Group, is an example of a virtual environment where the user can control his viewpoint and motion in the environment (Lippman, 1980). The Movie Map enables the operator to sit in front of a touch-sensitive display screen and drive through the town of Aspen, Colorado, at his own rate, taking any route he chooses, by touching the screen, indicating the turns he wants to make and the buildings he wants to enter.

Virtual environments have led to the concept of *telepresence*. This concept allows a user to feel as though he is in a real remote environment and can interact with it. This concept was developed by the Philco Corporation in 1958. The resulting system was a cathode-ray tube (CRT) mounted on the head of an operator. The CRT displayed images from a remote camera, and the operator was able to control the camera's viewpoint with his head (Comeau & Bryan, 1961).

Another virtual environment, also called virtual reality, includes a data glove and three-dimensional displays. The DataGlove™ was developed by Tom Zimmerman and introduced by VPL Research in 1987. Scott Fisher designed the first assembly of the DataGlove with a helmet-mounted display initially developed by Michael McGreevy at NASA Ames Research Center (Fisher, 1986). The helmet-mounted display included wide-angle binocular lenses connected to two tiny liquid-crystal display screens (Figure C.1). On these screens, a remote computer displays images of an environment (a robotic environment). The images on the two screens differ slightly, by a displacement of two and a half inches, as do the two scenes beamed simultaneously onto the two eyes in real life. Thus a person wearing the helmet-mounted display has binocular vision in the virtual environment.

The main laboratories involved in virtual environments are the Massachusetts Institute of Technology (MIT), NASA Ames Research Center, the Naval Ocean Systems Center in Hawaii, the University of North Carolina (Brooks, 1988, 1990), the University of Utah (Jacobsen *et al.*, 1984, 1990) and MITI's Tele-Existence Project in Japan.

C.2.1. The DataGlove

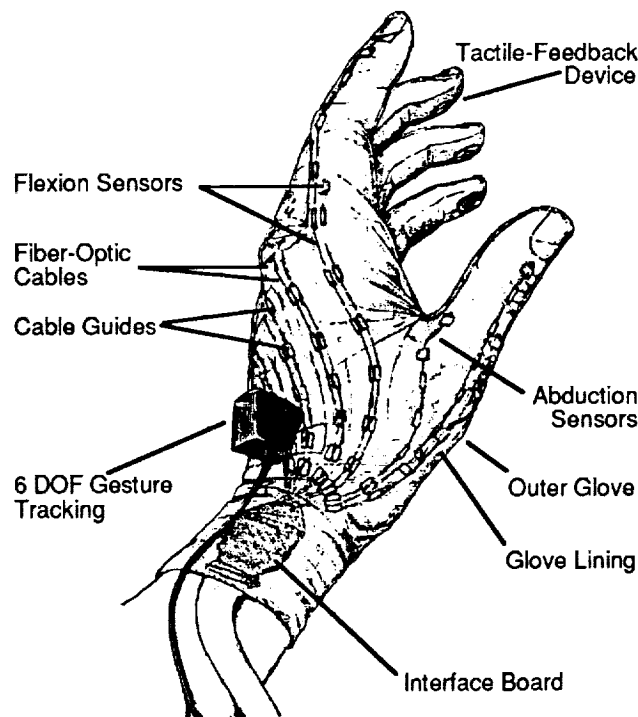


Figure C.2. Tactile input glove illustrating flex sensors.

The *DataGlove* is a light-weight, glove-like device that electronically records and transmits data records of hand and finger shape and dynamics to a host computer by measuring the amount of joint bend, finger abduction and thumb circumduction (Zimmerman *et al.*, 1986). The VPL Research system is based on an optical flex sensor technology. Fifteen flex sensors are mounted on a stretchable inner glove and are positioned lengthwise along the first and second joints of each finger, between each finger, between the thumb and first finger, and across the palm of each glove from thumb to fourth finger (Figure C.2).

The instrumented left and right inner gloves are covered by light-weight cotton outer gloves and are connected by nine conductor cables to a custom interface unit. To obtain information about the position and orientation of the hand and arm in the task environment, an additional six-degree-of-freedom tracking device is mounted above the wrist on the back of each DataGlove. The x, y, z, azimuth, elevation and roll coordinates of the hand are returned to the host system at up to 60 hertz. The DataGlove provides over 21 degrees of freedom for each hand of the human operator. Currently the gloves are instrumented with fibre-optic flex-sensing devices at each finger joint and between the fingers (Fisher *et al.*, 1988). This technology is useful for picking-up and manipulating virtual objects displayed in 3D in the virtual environment (see Section 3.2).

C.2.2. The Virtual Environment Workstation

The Aerospace Human Factors Division of NASA Ames Research Center developed an interactive virtual interface environment workstation (VIEW) (Fisher *et al.*, 1988) after a first attempt, called VIVED. VIEW provides a virtual auditory and stereoscopic image surround that is responsive to inputs from the operator's position, voice and gestures. It allows the user to explore a 360 degree synthesized or remotely sensed environment and to interact physically with its components (Fisher, 1989). VIEW consists of a wide-angle stereoscopic display unit, glove-like devices for multiple-degree-of-freedom tactile input, connected speech recognition technology, gesture tracking devices, 3D auditory display and speech synthesis technology, and computer graphics and video image generating equipment (Figure C.3).

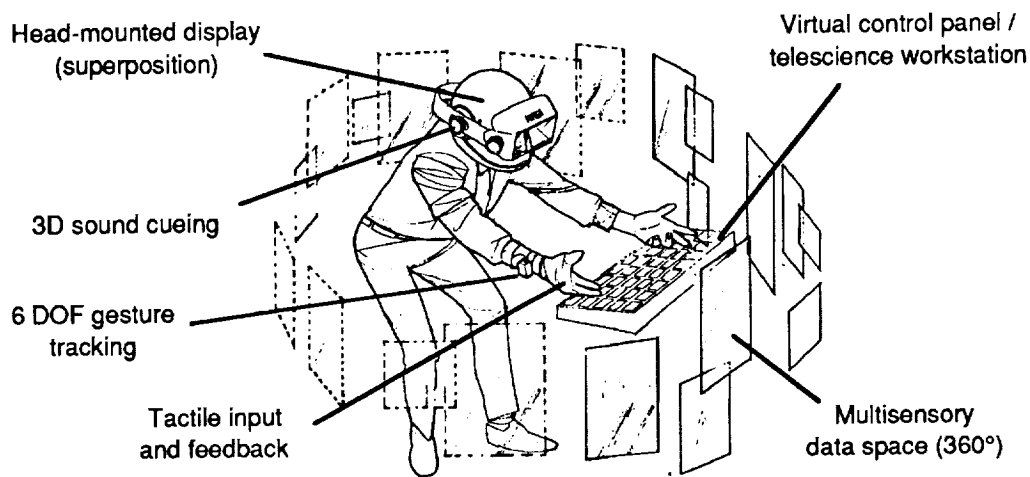


Figure C.3. Virtual interface environment workstation.

Head-mounted displays and optics

The 1988 version of the NASA head-mounted display unit used two passive, monochromatic, liquid-crystal display screens presented one to each eye of the user through wide-angle optics. Details are provided in Fisher *et al.* (1988). The optics provide a 120 degree horizontal and vertical field of view for each eye and up to a 90 degree binocular field overlap. The total instantaneous field of view is approximately 120 degrees. The displays and optics unit is positioned directly in front of the viewer's eyes and coupled

to head motion by means of a light-weight headgear configuration. A position tracking sensor, a microphone for input to the connected speech recognition subsystem, and earphones for auditory display feedback to the operator, are also available (Wenzel *et al.*, 1988).

3D displays

The computer image system allows high-performance, real-time 3D graphics presentation at resolutions of 620 times 220 pixel elements with 320 distinguishable vertical lines and approximatively 16 levels of greyscale. This imagery is generated at rates from 7 to 30 frames per second. For real-time video input of remote environments, two miniature CCD video cameras are used to provide stereoscopic imagery. In addition, a binaural auditory display is capable of presenting a wide variety of binaural sounds to the operator via earphones using sound synthesis technology developed for music synthesizers. The primary function of this auditory display is to provide both discrete and dynamic auditory cues which can augment or supply information missing from the visual or gestural displays.

C.2.3. Applications

Telerobotics and teleoperations

Virtual environments can be used to aid control of semiautonomous robots in remote and/or dangerous environments, e.g. proposed semiautonomous rovers on the Martian surface. Such robots have to communicate with humans and the type of communication involved will be remote and probably with long delays (Kim *et al.*, 1988). Such communication will probably be handled by operations procedures (plans), and these procedures have to be managed and maintained. They will constitute a high-level language shared between the semiautonomous robot(s) and the humans. They can be designed initially from design descriptions and rationale, but they have to be tailored to the tasks the robot has to accomplish. Procedure construction is an incremental process involving frequent feedback from the execution of various tasks.

One reasonable approach is case-based. The initial domain theory will come from design and will be modified incrementally by various eventualities encountered during operations. Operations have to be performed in an experimental environment first. The art of building procedures has to be understood from a cognitive point of view. Such understanding will help in real operations in the building of recovery strategies after unexpected events.

First, the robot must receive a signal to start its job (either preprogrammed or from a human). The robot then executes the corresponding procedure. If the procedure succeeds, the robot can start a subsequent task or wait for another command from the human. If the procedure does not succeed, the robot stops and sends the human a history of the events recorded during the preceding minutes. This could include the evolution of various essential control and environmental parameters, and, eventually, videotaped (or digitally encoded) scenes of the performed task. At this point, the human can replay the sequence of events leading up to the failure. This could be done in a virtual environment. Humans are very good at assessing situations when they are provided with realistic and relevant information. Clearly, they can also be helped by intelligent assistance if necessary. If the human operator detects a problem (e.g. an unexpected obstacle blocking the course of the robot), he can design a recovery procedure (in the virtual environment) to solve this unanticipated problem. At this point there are two types of solutions. Either the problem is already known (perhaps generic) and a recovery procedure has already been designed. In this case, the system can help in retrieving the corresponding procedure. Alternatively, the problem is new, and the human will have to invent an appropriate recovery procedure. This can be done in the virtual environment "by hand", and the movements recorded, interpreted

and transformed into a procedure understandable by the robot. In both cases, a procedure is generated which will be sent to the remote location where the real robot operates.

Medicine.

Another scenario in progress involves the development of a surgical simulator for medical students and plastic surgeons that could be used much as a flight simulators are used to train pilots. The surgeon can explore an electronic model of the body to do preoperation planning and patient analysis. Students can also see what the surgeon is currently looking at (Jacobsen *et al.*, 1984, 1990).

Appendix D

HyperCard and HyperTalk

HyperCard was developed by Bill Atkinson on the Macintosh computer at Apple. It is now part of the basic Macintosh software. It is so easy to learn to use that it is not even necessary to have any background in computer science to start using it. It provides the following capabilities: rapid prototyping and design of user interfaces; assembly and organization of text, graphics, images and sounds; segregation of large quantities of information into predefined units; and efficient manipulation of the units by the design of appropriate links between them. HyperCard version 2.0 allows display of multiple windows on the Macintosh screen.

HyperTalk is the underlying language of HyperCard and is based on the writing of *scripts*, sequences of instructions. HyperTalk has a syntax which is very easy to use. It allows event-driven or message-driven programming which is defined as follows.

The basic entities in HyperTalk are handlers. An *on handler* is a sort of reflex or DEMON. It reacts when a message is sent either by the mouse or by another part of HyperTalk. For instance, the handler "on myAction ..." is executed if the message "myAction" is sent to it. The "mouseUp" message is sent each time a mouse click is performed by the user. A *function handler* in HyperTalk is equivalent to a function in any other programming language.

In HyperCard, the predefined units are buttons, fields, cards, backgrounds and stacks. They all have properties and scripts attached and can be defined as objects. We will call the *basic objects* of HyperCard the buttons and the fields. *Buttons* are active areas on the HyperCard window and can be triggered by the mouse or by a message coming from any script of HyperCard. *Fields* are areas on the HyperCard window which contain text and within which it is possible to edit, select, cut, copy and paste any chunk of text. The *background* includes the buttons and fields common to a set of cards. Conversely, the *card* includes basic objects and information that may change. For instance, if you want to build a calendar, you write the day names, etc., that remain constant on the background and the things you have to do on a particular day will be written on the corresponding card. Basic objects in backgrounds and cards can be overlaid. It is thus possible to build temporary or permanent masks. A *stack* is the electronic equivalent to a card box. To maintain consistency, one stack is created for each homogeneous card. This is not, however, mandatory. Indeed, HyperCard allows the definition of as many backgrounds as may be needed within a stack. Within HyperCard there is a built-in *hierarchy* of objects, including an *inheritance* mechanism (Figure D.1).

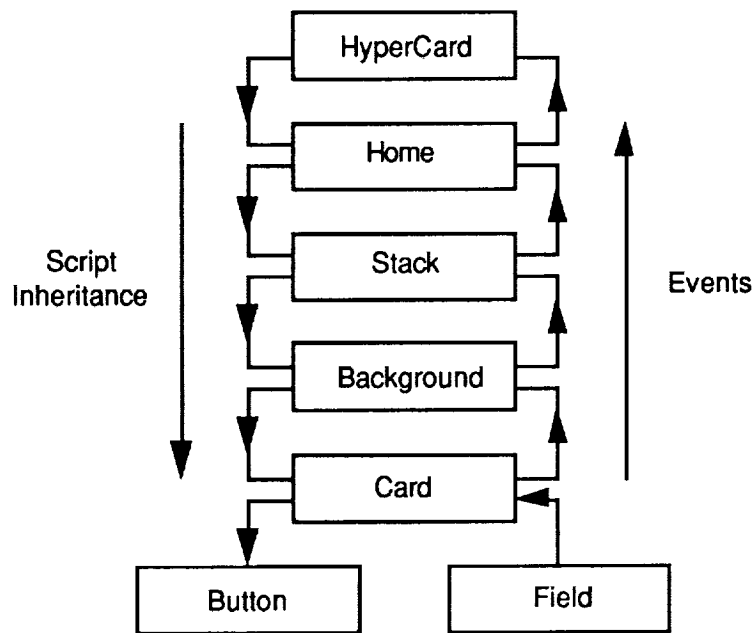


Figure D.1. Hierarchy in HyperCard.

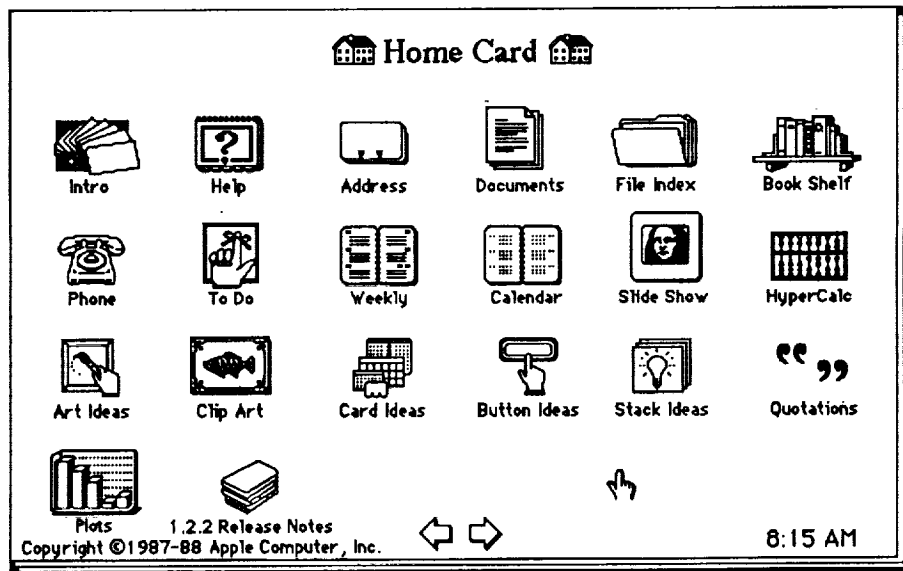


Figure D.2. The Home stack is necessary for HyperCard to work.

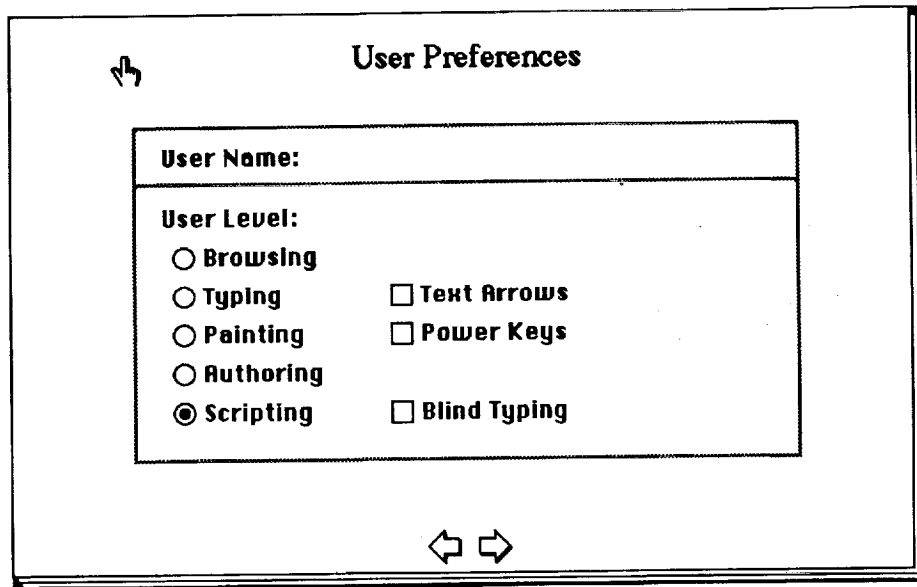


Figure D.3. User Preferences card in the Home stack.

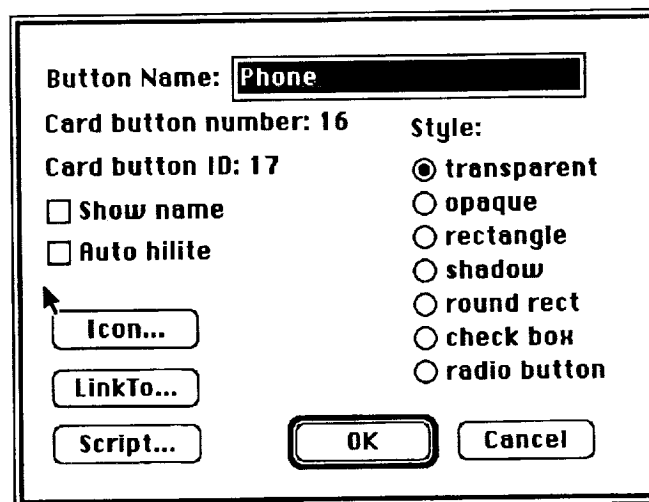


Figure D.4. Button properties.

There is a special stack called *Home* which is the first card the user sees the first time he launches a HyperCard application (Figure D.2). Home* plays the role of a desktop. An object inherits the scripts defined at a higher level, except if the same script is defined in its own script. Thus, this feature allows exceptions. As HyperCard is a message-driven system, when a message is sent, the first object of the hierarchy that includes a handler or a function describing this message intercepts the message.

HyperCard has five levels of usage corresponding to different types of users (Figure D.3). At the browsing level, which is a read-only mode, the user can browse through

* As a matter of fact, a HyperCard application can be built which will not display the Home card first. However, Home will still be on top of the stack in the hierarchy.

available stacks. At the typing level, the user can type text into fields. At the painting level, the user can add graphics to the stacks, either directly by using HyperCard tools or by importing MacPaint graphics. At the authoring level, the user can modify the structure of stacks. At the scripting level, the user has full control of the stacks.

A button has properties and a script which attach a behaviour to it. Properties include the number of the button in the card, its identification number, its style, its icon and several other options. As an example, the properties of the "Phone" button, displayed on Figure D.2, are presented in Figure D.4. The script of this button is presented in Figure D.5.

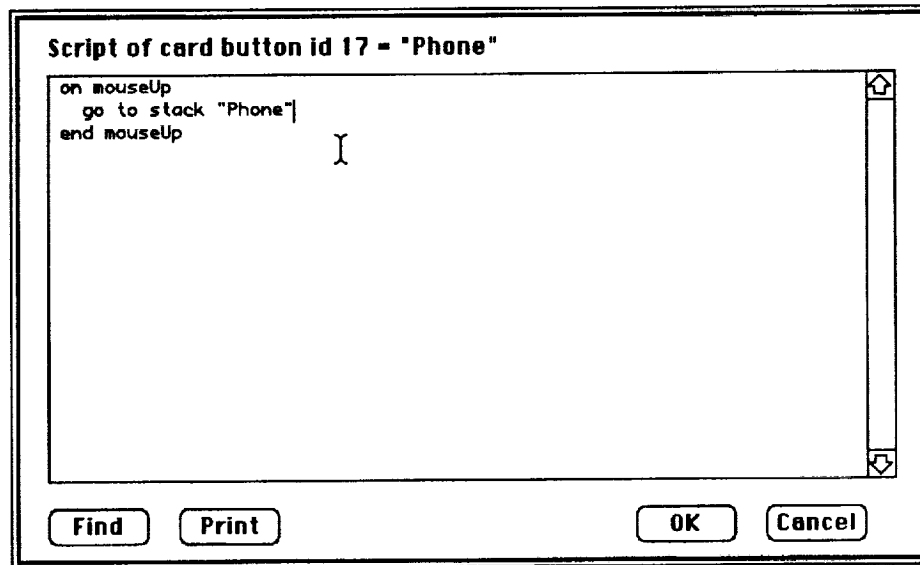


Figure D.5. Example of a script window.

The script window presented in Figure D.5 is the same as script windows for stacks, backgrounds, cards and fields. The script presented in this window says that when the mouse is up then the card called "Phone" will be displayed, i.e. the link from this button to the card "Phone" is activated. Scripts introduce a great deal of flexibility and provide a very interesting way to program dynamic links. Here, a *dynamic link* is a link between two nodes that is computed by a script. Since HyperTalk is an interpreted language, HyperCard applications may be very slow. For this reason, HyperCard provides a very interesting feature – external commands (XCMD) and external functions (XFCN). These functions can be written in C language or Pascal, compiled and attached as resources to HyperCard stacks. Such features speed up heavy calculations and provide a simple way of connecting HyperCard to the external world.

Appendix E

Abbreviations and Acronyms

AI	Artificial intelligence
AIM	Advanced Interaction Media
BID	Built-in descriptors
CDI	Compact disc interactive
CD ROM	Compact disc read-only memory
CERT	Centre d'Etudes et de Recherches de Toulouse
CID	Computer Integrated Documentation
CIDCP	CID control panel
CNES	Centre National d'Etudes Spatiales
CRT	Cathode-ray tube
CSCW	Computer supported cooperative work
DBMS	Database management system
HI	Human Intelligence
IAS	Intelligent assistant systems
IBIS	Issue-based information systems
IHMS	Integrated human-machine systems
ISO	International Organization for Standardization
JSC	NASA Johnson Space Center
KR	Knowledge representation
MIT	Massachusetts Institute of Technology
NASA	National Aeronautics and Space Administration
ORS	Orbital Refueling System
PALS	Program Automated Library System
PARC	Xerox Palo Alto Research Center
PC	Personal Computer
PDRD	Program Definition and Requirement Document
PRD	Program Requirement Document
PRS	Procedural Reasoning System
SRAR	Situation Recognition and Analytical Reasoning model
SRI	Stanford Research Institute
SSE	Software Support Environment
SSF	Space Station Freedom
TMIS	Technical Information Management System
VIEW	Virtual interface environment workstation
XCMD	HyperCard external command written in C or Pascal
XFCN	HyperCard external function written in C or Pascal

References

- Agre, P., & Chapman, D., (1987). Pengi: An Implementation of a Theory of Activity. *Proceedings of AAAI-87*, Morgan Kaufmann Publishers, San Mateo, CA.
- Anderson, J.R., (1976). *Language, Memory and Thought*. Hillsdale, Erlbaum, NJ.
- Anderson, J.R., (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.
- Ase, H. & Kobayashi, S. (1990). Information Retrieval Condition Generation System using Case-Based Reasoning. *Proceedings of Pacific Rim international Conference on Artificial Intelligence'90*, Nagoya, Japan, November 14-16.
- Asker, J.R. (1990). GAO Faults NASA for Mismanaging Storage of Valuable U.S. Space Science Data. *Aviation Week & Space Technology*, April 2.
- Aubron, S. & Hooper, K. (1988). *Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers*. Microsoft Press.
- Boy, G.A. (1986). An Expert System for Fault Diagnosis in Orbital Refueling Operations. *AIAA 24th Aerospace Sciences Meeting*, Jan., Reno, Nevada.
- Boy, G.A. (1987). Operator Assistant Systems. *Int. J. Man-Machine Studies*, 27, pp. 541-554.
- Boy, G.A. & Rappaport, A. (1987). Operator Assistant System in Space Telemannipulation: Knowledge Acquisition by Experimentation. *ROBEX*, Pittsburgh, USA, Ju. 4-5.
- Boy, G.A. & Delail, M. (1988). Knowledge Acquisition by Specialization-Structuring: A Space Telemannipulation Application. *AAAI-88, Workshop on Integration of Knowledge Acquisition and Performance Systems*, St Paul, Minnesota, USA.
- Boy, G.A., & Mathé, N., (1989). Operator Assistant Systems: An Experimental Approach Using A Telerobotics Application. *IJCAI Workshop on Integrated Human-Machine Intelligence in Aerospace Systems*, Detroit, Michigan, USA, August 21.
- Boy, G.A. (1989). The Block Representation in Knowledge Acquisition for Computer Integrated Documentation. *Proceedings Knowledge Acquisition for Knowledge-Based Systems, AAAI Workshop*, Banff, Canada.
- Boy, G.A. (1991). *Intelligent Assistant Systems*. Academic Press, London.

- Brand, S. (1987). *The Media Lab: Inventing the Future at MIT*. Viking Penguin.
- Brown, P.J. (1989). Linking and Searching within Hypertext. *Electronic Publishing-Origin, Dissemination and Design 1*, 1, April, pp. 45-53.
- Brown, P.J. (1989). Do we need maps to navigate round hypertext documents? *Electronic Publishing-Origin, Dissemination and Design 2*, 2, July, pp. 91-100.
- Canter, D., Rivers, R. & Storrs, G. (1985). Characterizing User Navigation through Complex Data Structures. *Behaviour and Information technology* 4, 2, April-June, pp. 93-102.
- Carbonell, J.G. & Hood, G. (1986). The World Modelers Project: Learning in a Reactive Environment. in Mitchell, T.M., Carbonell, J.G., & Michalski, R.S. (editors), *Machine Learning: A Guide to Current Research*, Kluwer Academic Press, pp. 29-34.
- Chase, W.G. & Simon, H.A. (1973). Perception in Chess. *Cognitive Psychology*, 4, pp. 55-81.
- Chen, H. & Dhar, V. (1987). Reducing Indeterminism in Consultation: A Cognitive Model of User/Librarian Interactions. Proceedings of AAAI'87, Seattle, WA.
- Cochrane, P.A. & Markey, K. (1985). Preparing for the Use of Classification in Online Cataloging Systems and in Online Catalogs. *Information Technology and Libraries*, 4, 2, pp. 91-111.
- Conklin, J. (1987). Hypertext: An Introduction and Survey. Computer, September.
- Conklin, J. & Begeman, M.L. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions Office Information Systems* 6, 4, October, pp. 303-331.
- Croft, W.B. & Turtle, H. (1989). A Retrieval Model for Incorporating Hypertext Links. *Hypertext'89 Proceedings*, pp. 213-224, ACM press, New York, 1989.
- Crouch, D.B., Crouch, C.J. & Andreas, G. (1989). The Use of Cluster Hierarchies in Hypertext Information Retrieval. *Hypertext'89 Proceedings*, pp. 225-237, ACM press, New York, 1989.
- Dede, C.J., Sullivan, T.R. & Scace, J.L. (1988). *Factors Shaping the Evolution of Electronic Documentation Systems*. Technical report, Research Activity No. IM. 4, Research Institute for Computing and Information Systems, University of Houston, Clear Lake.
- deKleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28, pp. 127-162.
- Dreyfus, H.L. (1979). *What Computer Can't Do, The Limits of Artificial Intelligence*. Harper and Row, Pub., Inc., New York.
- Doszko, T.E. (1983). CITE NLM: Natural Language Searching in an Online Catalog. *Information Technology and Libraries*, 2, 4, pp. 364-380.
- Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12, pp. 231-272.

Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S. & Harshman, R. (1988). Using Latent Semantic Indexing to Improve Access to Textual Information. *Proceedings of the ACM CHI'88*, Washington D.C., May 15-19, pp. 281-285.

Egan, D.E., Remde, J.R., Landauer, T.K., Lochbaum, C.C. & Gomez, L.M. (1989). Acquiring Information in Books and SuperBooks. *Proceedings of the Annual Meeting of the American Educational Research Association*, San Francisco, CA, March 27-30.

Engelbart, D.C. (1963). A Conceptual Framework for the Augmentation of Man's Intellect. In *Vistas in Information Handling*, Vol. 1, Spartan Books, London.

Falzon, P. (1986). La communication Homme-Machine: Les Strategies de Dialogue et le Langage de l'Interaction. *Proc. Workshop on Human Machine Interaction and Artificial Intelligence in Aeronautics and Space*, G. Boy Ed., ENSAE/CERT, Toulouse, Oct. 13-14.

Falzon, P. & Visser, W. (1989). Variation in Expertise. Implications for the Design of Assistance Systems. *Third International Conference on Human-Computer Interaction*, Boston, Elsevier Sc. Publishers.

Fikes, P.E. & Nilsson, N.J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2, pp. 189-208.

Fiscal Year 1985 Congressional Budget. (1985). *Cost Per Flight Operations Costs*. Office of Management and Budget.

Fischer, G. & Stevens, C. (1990). *Information Access in Complex, Poorly Structured Information Spaces*. University of Colorado at Boulder. Technical Report CU-CS-461-90, February.

Fischer, G., McCall, R. & Morch, A. (1989). Design Environments for Constructive and Argumentative Design. *Proceedings of the ACM CHI'89*, Austin, TX, April 30-May 4, pp. 269-275.

Fischer, G., Henninger, S. & Redmiles, D. (1990). A Conceptual Framework and Innovative Systems for Accessing Knowledge for Software Reuse. Draft of a paper to be presented at the *HCI Consortium 1990 Winter Conference*, February.

Fisher, D.H. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 2, pp. 139-172.

Foss, C.L. (1988). Effective Browsing in Hypertext Systems. *Proceedings of RIAO'88 Conference User-Oriented Context-Based Text and Image Handling*, MIT, Cambridge, MA, March 21-24, pp. 82-98.

Friedland, P.E. (1981). Acquisition of procedural knowledge from domain experts. *Proceedings of the Seventh International Joint Conference in AI*, Vancouver, B.C., Canada, pp.856-861.

Genesereth, M.R. & Nilsson, N.J. (1987). *Logical Foundation of Artificial Intelligence*. Morgan Kaufmann Publisher, Inc., Los Altos, CA.

Gruber, T. (1989). Automated Knowledge Acquisition for Strategic Knowledge. Report no. KSL 89-12, Knowledge System Laboratory, Computer Science Department, Stanford

University, Stanford, CA 94305, to appear in a special issue of *Machine Learning* on automated knowledge acquisition.

Harris, L.R. (1977). User Oriented Data BaseQuery with the ROBOT Natural Language Query System. *International Journal of Man Machine Studies*, 9, pp. 697-713.

Hayes-Roth, B., Washington, R., Hewett, R & Hewett, M. (1989). Intelligent Monitoring and Control. *Proceedings of IJCAI*, Detroit, Michigan.

Hjerppe, R.. Project HYPERCATalog: Vision and Preliminary Conceptions of an Extended and Enhanced Catalog. *Proceedings of 6th IRFIS*, Frascati, Italy, pp. 15-18.

Humphrey, S.M. (1987). Illustrated Description of an Interactive Knowledge-Based Indexing System. *Proceedings of the 10th International ACM SIGIR Conference on R&D in Information Retrieval*. New Orleans. ACM Press, New York, pp. 73-90.

Jones, R. (1989). *A Model of Retrieval in Problem Solving*. PhD Dissertation. Department of Information and Computer Science, University of California, Irvine, CA.

Jones, W.P. (1987). How do We Distinguish the Hyper from the Hype in Non-Linear Text ? *Human-Computer Interaction - INTERACT'87*, H.J. Bullinger and B. Shackel (Editors), Elsevier Science Publishers, North Holland.

Korf, R.E. (1985). Macro-Operators: A Weak Method for Learning. *Artificial Intelligence*, 26, pp.35-77.

Kibby, M.R. & Mayes, J.T. (1989). Towards Intelligent Hypertext. In McAleese, R. (Ed.) *Hypertext Theory into Practice*, Albex 1989, pp. 164-172.

Laird, J.E., Newell, A. & Rosenbloom, P.S. (1984). Towards chunking as a general mechanism. *Proceedings AAAI-84*, Austin, Texas.

Laird, J.E., Newell, A. & Rosenbloom, P.S. (1985). Soar: An Architecture for General Intelligence. *Artificial Intelligence*, .

Laird, J.E., Rosenbloom, P.S. & Newell, A. (1986). Chunking in SOAR: The Anatomy of a General Mechanism. *Machine Learning*, Kluwer Academic Publisher, The Netherlands.

Lowry, D.J. & Feaster, T.A. (1987). Regarding Space Leadership through Control of Life Cycle Costs. *25th International Space Conference*, Cocoa Beach, Florida.

Luhn, H.P. (1957). A Statistical Approach to the Mechanized Encoding and Searching of Literary Information. *IBM Journal of Documentation*, 28:1, October, pp. 309-317.

MacLean, A., Carter, K., Lövstrand L. & Moran, T. (1990). User-Tailorable Systems: Pressing the Issues with Buttons. *Proceedings of the ACM CHI'90*, Seattle, Washington, April 1-5.

Martin, J. (1990). *Hyperdocuments and How to Create Them*. Prentice Hall, Englewood Cliffs, NJ.

Mathé, M. (1990). *Intelligent Assistance to Process Control: A Space Telemanipulation Application* (In French). PhD Thesis Dissertation. ENSAE, Toulouse, France.

- Miller, G.A. (1956). The magic number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63, pp. 81-97.
- Minsky, M. (1985). *Society of Minds*. Touchstone Books, Published by Simon & Schuster Inc., New York.
- Minsky, M. (1975). A Framework for Representing Knowledge. In P. Winston Ed., *The Psychology of Computer Vision*, 211-277, New York: McGraw Hill.
- Mitchell, T.M. (1982). Generalization as Search. *Artificial Intelligence*, 18, pp. 203-226.
- Monk, A. (1989). The Personal Browser: A Tool for Directed Navigation in Hypertext Systems. *Interacting with Computers* 1, 2, August, pp. 190-196.
- Nelson, T.H. (1967). Getting It Out of Our System. *Information Retrieval: A Critical Review*, G. Schechter, Ed., Thompson Books, Washington D.C.
- Newell, A., & Simon, H., (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, N.J.
- Newell, A. & Simon, H. (1981). Mechanisms of Skill Acquisition and the Law of Practice. In J.R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Lawrence Erlbaum, Hillsdale, NJ.
- Nielsen, J. (1989). *HyperText & HyperMedia*. Academic Press, London, UK.
- Nilsson, N.J. (1988). Action Networks. Computer Science Department, Stanford University, Stanford, CA 94305.
- Oatley, K.G. (1977). Inference, Navigation, and Cognitive Maps. In Johnson-Laird, P.N. & Wason, P. (Eds.) *Thinking, Readings in Cognitive Science*, Open University Set Book, Cambridge University Press, Cambridge, pp. 537-547.
- Ochanine, D.A. (1981). L'Image Operative. *Actes d'un séminaire et recueil d'articles, Centre d'Education Permanente, Département d'Ergonomie et Ecologie Humaine, Université Paris I*.
- Ohlsson, S. (1987). Transfer of Training in Procedural Learning: A Matter of Conjectures and Refutations? In L. Bolc (Ed.), *Computational Models of Learning*. Springer Verlag, Berlin.
- Parsaye, K., Chignell, M., Khoshafian, S. & Wong, H. (1989). *Intelligent Databases - Object-Oriented, Deductive Hypermedia Technologies*. Wiley, New York.
- Pollitt, A.S. (1989). *Information Storage and Retrieval Systems, Origin, Development and Applications*. Ellis Horwood Limited, Publishers, Chichester, John Wiley & Sons, New York.
- Quillian, M.R. (1968). Semantic Memory. In M.L. Minsky (Ed.), *Semantic Information Processing*. MIT Press, Cambridge, MA.

Rappaport, A.R. & Gaines, B. (1988). Integration of Acquisition and Performance Systems. *Proceedings of the Integration of Knowledge Acquisition and Performance Systems Workshop*, AAAI-88, St Paul, Minnesota, August 21.

Reason, J. (1986). Decision Aids: Prothesis or Tools ? *Intelligent Decision Support in Process Environments*, NATO Workshop, Nov. 11-14, Ispra, Italy.

Quillian, M.R. (1968). Semantic Memory. In M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, MA, pp. 216-260.

Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. North-Holland, New York.

Rogers, W.P. (1986). *Presidential Commission on the Space Shuttle Challenger Accident Report*. U.S. Government Printing Office: 688-274/58381, June.

Salton, G. (1989). *Automatic Text Processing. The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, Reading.

Schank, R.C. & Abelson, R.P. (1977). *Scripts, Plans, Goals and Understanding*. Laurence Erlbaum, Hillsdale, N.J., pp. 1-68.

Shell, P. & Carbonell, J.G. (1989). Towards a General framework for Composing Disjunctive and Iterative Macro-Operators. *Proceeding of IJCAI*, Detroit, Michigan.

Spark Jones, K. (1972). A Statistical Interpretation of Term Specificity and its Application in Retrieval. *Journal of Documentation*, 28:1, March, pp. 11-21.

Stotts, P.D. & Futura, R. (1988). Adding Browsing Semantics to the Hypertext Model. *Proceedings of the ACM Conference on Document Processing Systems*, Santa Fe, NM, December 5-9, pp. 43-50.

Tambe, M. & Newell, A. (1988). *Why Some Chunks are Expensive*. CMU-CS-88-103, Department of Computer Science, Carnegie Mellon University, Pittsburgh.

Trigg, R.H. & Weister, M. (1986). Text Net: A Network Based Approach to Text Handling. *ACM Transactions on Office Information Systems*, 4:1, January, pp. 1-23.

Walker, D. (1981). The Organization and Use of Information: Contributions of Information Science, Computational Linguistics and Artificial Intelligence. *Journal of the American Society for Information Science*, Sept., pp. 347-363.

Waltz, D.L. (1978). An English Language Question Answering System for a Large Relational Data Base. *Communications of the ACM*, 21, pp. 526-539.

Weyer, S.A. (1988). As we May Learn. In Aubron, S. & Hooper, K. (1988). *Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers*. Microsoft Press, pp. 87-103.

Whalen, T. & Patrick, A. (1989). Conversational Hypertext: INformation Access through Natural Language Dialogues with Computers. *Proceedings of the ACM CHI'89*, Austin, TX, April 30-May 4, pp. 289-292.

Williamson, K.E., (1986), "Learning from Exceptions in Databases", in *Machine Learning: A Guide to Current Research*, edited by Tom M. Mitchell, Jaime G. Carbonell, and Ryszard S. Michalski, Kluwer Academic Publishers.

Woods, W.A. (1975). What's in a Link: Foundation for Semantic Networks. in Bobrow D.G., Colins A. (eds.), *Representations and Understanding: Studies in Cognitive Science*, Academic Press, New York, pp. 35-84.

Yankelovich, N., Meyrowitz, N. & van Dam, A. (1985). Reading and Writing the Electronic Book. *Computer*, 18:10, October, pp. 15-30.

Yu, C.T., Luk, W.S. & Siu, M.K. (1979). On Models of Information Retrieval Processes. *Information Systems*, 4:3, pp. 205-218.

Zimmerman, M., (1988), "TEX version 0.5", Silver Spring, MD.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1991		3. REPORT TYPE AND DATES COVERED Technical Memorandum
4. TITLE AND SUBTITLE Computer Integrated Documentation			5. FUNDING NUMBERS RTOP 590-12-12 NAS 2-13210	
6. AUTHOR(S) Guy Boy				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ames Research Center Moffett Field, CA 94035-1000			8. PERFORMING ORGANIZATION REPORT NUMBER A-91167	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001			10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-103870	
11. SUPPLEMENTARY NOTES Point of Contact: Guy Boy, Ames Research Center, MS 244-17, Moffett Field, CA 94035-1000; (415) 604-3369 or FTS 464-3369				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified — Unlimited Working Paper Subject Category 61			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This technical memorandum (TM) presents the main technical issues of the Computer Integrated Documentation (CID) project. The problem of automation of documents management and maintenance is analyzed both from an artificial intelligence viewpoint and from a human factors viewpoint. Possible technologies for CID are reviewed: conventional approaches to indexing and information retrieval, hypertext, and knowledge-based systems. A particular effort has been made to provide an appropriate representation for contextual knowledge. This representation is used to generate context on hypertext links. Thus, indexing in CID is context-sensitive. The implementation of the current version of CID is described. It includes a hypertext database, a knowledge-based management and maintenance system, and a user interface. This TM also provides a series of theoretical considerations as navigation in hyperspace, acquisition of indexing knowledge, generation and maintenance of a large documentation, and relation to other work.				
14. SUBJECT TERMS Hypermedia, Information retrieval, Context-sensitive indexing, Artificial intelligence, Human-computer interaction			15. NUMBER OF PAGES 132	
			16. PRICE CODE A07	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	